

AFRL-IF-RS-TR-2006-14
In-House Interim Report
January 2006



HYBRID ARCHITECTURES FOR EVOLUTIONARY COMPUTING ALGORITHMS

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2006-14 has been reviewed and is approved for publication

APPROVED: /s/

DANIEL BURNS
Project Engineer

FOR THE DIRECTOR: /s/

JAMES A. COLLINS
Deputy Chief, Advanced Computing Division
Information Directorate

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> <i>OMB No. 074-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE JANUARY 2006	3. REPORT TYPE AND DATES COVERED In-House Interim, Mar 2003 – Mar 2005	
4. TITLE AND SUBTITLE HYBRID ARCHITECTURES FOR EVOLUTIONARY COMPUTING ALGORITHMS			5. FUNDING NUMBERS C - N/A PE - 62702F PR - 459T TA - HA WU - EC	
6. AUTHOR(S) Daniel Burns				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFTC 525 Brooks Road Rome New York 13441-4505			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFTC 525 Brooks Road Rome New York 13441-4505			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2006-14	
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Daniel Burns /IFTC/(315) 330-2335/ Daniel.Burns@rl.af.mil				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 Words) This report documents interim progress for an in-house project aimed at identifying, developing and evaluating applications of evolutionary computing methods to hard optimization problem test cases on a single PC computer, a cluster of computers, and hardware FPGA platforms. We surveyed evolutionary computing literature and chose to focus on the Generic Algorithm, GA. We had the GA test three case problems, Non-Linear Coupled Ordinary Differential Equation, ODE, Parameterization, the DNA Code Word Library Generation, and the Networked Senior Power Management Policy Problem. The first test problem used an ODE bio-model for Antigen-Antibody binding that was of interest to a PI for a DARPA SIMBIOSYS program we managed. We developed prototype optimization software tools in three programming environments, Labview, Matlab, and compiled C, and demonstrated speed-ups on the order of 100-1000x by moving to C. We parallelized the C codes using Message Passing Interface and demonstrated good linear speed-ups on a cluster. Our GA solution for the second test case problem, DNA Code Word Library Generation, was also parallelized, and was faster than any algorithm found in the literature. Finally, we began developing a hardware accelerated version of GA for the DNA Code Word Problem as a first step toward a distributed hardware implementation.				
14. SUBJECT TERMS Genetic Algorithm, Optimization, Ordinary Differential Equation, parallel, distributed, Field Programmable Logic Array				15. NUMBER OF PAGES 51
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Table of Contents

Project Goals	1
Accomplishments from 4/01/03 to 8/19/05	1
Tasks to be addressed in next period	2
Publications/Briefings During this Period	2
Description of Accomplishments.....	3
Spiral 1: (PC platform).....	3
Literature Search, State-of-the-Art, Gap Analysis.....	3
Labview version of GA optimizer and Ag/Ab binding bio-model ported to C.....	5
MatLab GA and bio-models	10
MatLab v2 bio-model ported to C	11
Evaluation of speed, accuracy, convergence, and scaling performances of the Labview, MatLab, and compiled C versions	11
Evaluation of GEPASI Bio-model Simulator	20
Java/OAA wrapped PC GA ODE Parameterizer version prepared as BioSpice agent.	20
Web Browser Interface PC GA ODE Parameterizer Version	21
Summary of work in Spiral 1 (PC platform)	22
Spiral 2: (Cluster platform).....	22
GA ODE Parameterization Problem.....	22
Island Model distributed GA	24
DNA Code Word Library Generation Problem.....	25
Application of distributed GA to Sensor Network Energy Management:	28
Summary of work in Spiral 2 (Cluster platform).....	28
Spiral 3 (Hardware Accelerated Platforms).....	28
Summary of work in Spiral 3.....	34
Remaining work in Spiral 3	34
Future work.....	34
Acknowledgements:.....	37

List of Appendices

Appendix A. GECCO 2004 Paper	39
Appendix B. FNANO 2005 Paper	42
Appendix C. GECCO 2005 Paper	44

List of Figures

Figure 1: Front Panel of GA optimizer tool solver in Labview version.	6
Figure 2: Structure of Epitope-Paratope Bond Formation Model	7
Figure 3: Two Compartment Model Incorporating Steric Hindrances.....	7
Figure 4: Front panel of equation ODE solver in Labview version.....	8
Figure 5: Speed Test_0, solution time of the initial population.....	12
as a function of the number of individuals in the initial population.	12

Figure 6: Speed Test_1, solution time of the target population.	13
Figure 7: Elapsed time for fitting task for the versions.	14
Figure 8: Average maximum parameter fitting error for the versions.	15
Figure 9: Average number of generations for good fit for the versions.	16
Figure 10: Elapsed time and Sum Err fitting different numbers of parameters (Cv1 version)....	16
Figure 11: Effect of parameter search range on performance.....	17
Figure 12: Effect of fitting with relaxed accuracy requirement.....	18
Figure 13a. Fitting all data points, no noise.....	19
Figure 13c. Fitting every 10th data point, no noise.	19
Figure 13b. Fitting all data points, 20% random noise.....	19
Figure 13d. Fitting every 10th data point, 20% random noise.....	19
Figure 14: Outline of web browser interface for GA ODE Parameterizer.	22
Figure 15: Jumpshot time profiling of Farming Model distributed GA (26 processors, beyond speed-up plateau).	23
Figure 16: Speed-up curve for best tuning of the Farming Model distributed GA.....	24
Figure 17: Speed-up curve for distributed Island Model GA ODE Parameterizer.....	25
Figure 18: Multiple run data analysis tool front panel display.	26
Figure 19: Comparison of Markov, GA, and stochastic DNA Code Word Library Generation methods.	27
Figure 20: Levenstein Matrix Calculation	30
Figure 21: Levenstein matrix calculation array implementation.	31
Figure 22: Upper level functional block diagram of fitness evaluator.	32
Figure 23: Maximum numbers that can be calculated in Levenstein matrix cells.....	33
Figure 24: Higher level function block diagram of GA optimization FPGA hardware core.	34
Figure 25: Speed-up and resources for the various platforms considered by this project.	36

List of Tables

Table 1 Invited Speakers, IF EC Interest Group.....	4
Table 2 Primary GA optimizer parameters.....	6
Table 3 Primary Bio-model parameters.....	9
Table 4 Characteristics of various GA fitter/bio-model versions.	11
Table 5 Performance and Scaling metrics for GA bio-model fitter.....	11
Table 6 % of Runs Reaching Maximum of Generations	14
Table 7 Parameters being fit for sets of bars in Figure 9.	17
Table 8 Time profiling study of GA/DNA Code Word Library Generation application showing time consumed by subroutine (produced with GNU gprof).	29
Table 9 Synthesis report showing resource utilization and expected speed. Minimum period: 12.37ns (80.8MHz)	32

Project Goals

This project will investigate novel computing architectures that facilitate evolutionary computing (EC) methods such as Genetic Algorithms (GAs) and Genetic Programming (GP) that are being applied by an increasing number of researchers to hard, NP-complete combinatorial optimization problems in a number of diverse problem domains. One goal is to determine whether EC based algorithms offer any advantages over more classical methods, especially in the context of parallel and hybrid (or heterogeneous) hardware/software implementations that are aimed at achieving extreme solution time speed-ups and problem size scaling. A focus of this work will be to evaluate the performance of these methods running on both modest conventional computer platforms as well as on a new heterogeneous computer that uses a field programmable logic array at (FPGA) each node in a cluster of computers. The results of this research may lead to improved solution engines for NP-complete optimization problems that are relevant to a number of IF mission area applications. Another goal is to develop new optimization tools for parameterizing and optimizing systems making use of bio-models relevant to current DARPA programs managed by AFRL/IF.

Accomplishments from 4/01/03 to 8/19/05

Spiral 1: (PC platform - complete)

- Translated Labview version GA optimization tool and Antigen/Antibody (Ag/Ab) binding bio-model to C
- Obtained MatLab based Genetic Algorithm Optimization Toolbox from North Carolina State University, integrated MatLab bio-models from Purdue University SIMBIOSYS PI and also with C version bio-model derived from Labview version.
- Ported Purdue MatLab bio-model to C, integrated with C version.
- Evaluated the speed, accuracy, convergence, and scaling performances of the Labview, MatLab, and C versions of GA Ordinary Differential Equation (ODE) Parameterization tool
- Evaluated Virginia Tech GEPASI bio-model simulation and fitting tool
- Developed Java Open Agent Architecture (OAA) wrapped compiled C version for contribution as BioSpice agent under the DARPA BIOCOMP program
- Developed web browser interface version for used by remote, no-programmer users
- Established Evolutionary Computing Interest Group at IF and hosted 7 speakers.

Spiral 2: (Cluster platform - complete)

- Developed Distributed Farming and Island Model GA applications to Non-Linear ODE parameterization (C/Message Passing Interface (MPI)), evaluated performance scaling vs. # processor nodes.
- Developed 2nd application of Distributed GA to DNA Code Word Library Generation problem and demonstrated linear speed-up performance scaling vs. # processors nodes.
- Developed 3rd application of GA to Networked Sensor Power Management Problem
- Visited AFIT, Wright State University, Virginia Tech. to discuss collaborations.

Spiral 3 (Hardware Accelerated Platforms)

- Collaborated with Impulse, Inc. evaluating Co_Developer C to VHDL translator tool
- Preliminary design completed for GA optimization algorithm FPGA core written in VHDL for hardware implementation aimed at extreme speed-up.
- Final design completed in VHDL for Levenstein Matrix systolic array calculator hardware accelerator for fitness function evaluation speed-up for DNA Code Word Library Generation Problem.

Tasks to be addressed in next period

- Complete GA core in VHDL for FPGA hardware implementation
- Integrate GA core and DNA Code Word Library generator fitness function evaluator in one FPGA hardware accelerator
- Evaluate and purchase hardware acceleration platforms for Notebook PC PCMCIA card, prototype GA DNA Code Word Library FPGA version, do notebook prototype.
- Evaluate and purchase FPGA board for 1 node of the IFTC G5 BIOCOMP cluster, prototype GA DNA Code Word Library FPGA, do prototype on that platform.
- Evaluate HHPC multiple FPGA version of GA DNA Code Word Library Generator.

Publications/Briefings During this Period

D.J. Burns and K.N. May, “On Parameterizing Models of Antigen-Antibody Binding Dynamics on Surfaces – a Genetic Algorithm Approach and the Need for Speed”, Proceedings of the Genetic and Evolutionary Computing Conference – GECCO 2004, Seattle, WA, June, 2004, Vol. 1, pp. 497-498. (Workshop and open poster session papers, see Appendix A).

D. J. Burns, K.N. May, and M. Bishop, “Parallel Genetic Algorithm for DNA Codeword Library Design”, 2nd Conference on Foundations of Nanoscience – Self-Assembled Architectures and Devices, Snowbird, Utah, Apr. 2005, pp. 128-129. (Poster paper, see Appendix B).

D.J. Burns, K.N. May, M. Bishop, “DNA Code Word Library Generation Using a Parallel Genetic Algorithm”, Workshop on Military and Security Applications of Evolutionary Computing, Genetic and Evolutionary Computing Conference – GECCO 2005, Washington, D.C., June, 2005. (see Appendix C).

K.N. May, “Genetic Algorithm Solvers for Non-Linear ODE Parameterization and DNA Code Word Library Generation on PC and Cluster Platforms”, Undergraduate Student Workshop, Genetic and Evolutionary Computing Conference – GECCO 2005, Washington, DC, June, 2005. (Invited presentation).

D. Burns, “Air Force Application of Evolutionary Computing Algorithms”, Genetic and Evolutionary Computing Conference – GECCO 2005, Washington, DC, June, 2005. (Regular session briefing, not published in Proceedings).

Briefed to 2003 Scientific Advisor Board in Advanced Computer Architecture during ACA focus area poster session. [\\Lfs-projects\Projects\SAB_2003\SAB_Focus_Area_Sessions\Advanced.Computing.Architectures\Poster.Session\ACA.30.Hyb_Arch_Evol_Comp_Meth.Burns.VF.ppt](#)

Description of Accomplishments

The work in this period has completed two of the three major tasks, i.e. identifying, developing and evaluating applications of evolutionary computing methods to test case problems run on both a *single PC computer* and on a *cluster of computers*. We also made a good start on the third spiral that will pursue further speed-ups by moving to *embedded hardware* implementations, a Field Programmable Logic Array (FPGA). In the first spiral we surveyed evolutionary computing literature and chose to focus on the Genetic Algorithm. We developed prototype optimization software tools in three programming environments (Labview, MatLab, and compiled C), and evaluating their relative performances solving a test case problem. The test case problem was parameterizing a particular bio-model consisting of a set of Non-Linear, coupled Ordinary Differential Equations (ODE). This problem is of wide interest to workers in certain biologically oriented DARPA program we are managing. In the second spiral we parallelized the C code version using MPI to run on a distributed cluster computer. We also applied the tool to solve two additional optimization problems of interest to others in IF, namely the DNA Code Word Library Generation Problem, and Networked Sensor Power Management Policy Problem, and we evaluated its performance relative to the best examples of other methods in the literature. Finally, in the third spiral, we began developing a hardware accelerated version of a GA optimizer as applied to the DNA Code Word Problem, as a first step toward a distributed hardware implementation.

Most of the work reported here was performed in-house at IFTC by Dan Burns and Kevin May (Clarkson Summer Engineering Aide summer 2003, 2004, 2005 and winter break 2003, 2004). The preliminary distributed Island Model GA and FPGA core design was assisted by Dr. Larry Merkle, Rose Hulman Institute of Technology (2004 Summer Visiting Faculty Research assignment in AFRL/IFTC with Mr. Burns), and the Networked Sensor Power Management Policy application was done by Dr. Qinru Qiu, Binghamton University (2005 Summer Visiting Faculty Research assignment in IFTC with by Mr. Burns).

The remainder of this report discusses the technical work accomplished during this period on the three main tasks, and Appendices follow with copies of publications generated by this project.

Spiral 1: (PC platform)

Literature Search, State-of-the-Art, Gap Analysis

A standard literature search was done which identified a large number of references on the development and use of Evolutionary Computing algorithms for various optimization problems in many domains. One of the largest and widest ranging meetings on the subject is the Genetic and Evolutionary Computing Conference, which Mr. Burns attended in 2003-2005. The

meeting includes separate days of tutorials and workshops in addition to the concurrent regular sessions, and these helped us to rapidly assess the state-of-the-art in this area. The 2004 and 2005 meetings also included a workshop specifically on Military and Security Applications of Evolutionary Computing. This workshop was organized by a leading worker in the field, David Goldberg/University of Illinois at Urbana-Champaign (UIUC), with support by AFOSR (Maj. J. Vasquez). Mr. Burns AFRL/IFTC and Ms. Misty Blowers AFRL/IFEC have since helped organize this workshop in 2004 and 2005, respectively. We also started an informal Evolutionary Computing Interest Group in IF that met at least 6 times to hear the invited speakers shown in Table 1, and to discuss both in-house and contractual applications of EC at IF. These speakers were sponsored by the Chief Scientist Lecture Series or the Information Institute, except Dr. Ridder, who was sponsored the Navy, and Maj. Vasquez, whose visit was sponsored by AFOSR. We also visited and consulted workers in the WPAFB area who have been working on EC topics, including with Prof. Gary Lamont/AFIT <http://en.afil.edu/hpc/evolution.html> , who has applied GA's to multi-objective optimization problems, and Dr. John Gallagher/Wright State University, whose interest is compact and mini-pop GA's http://carl.cs.wright.edu/ehrg/e_home/e_home.html .

Table 1 Invited Speakers, IF EC Interest Group

Speaker	Topic	Date
Dr. Kenneth De Jong, George Mason University	Taxonomy of Evolutionary Computing Methods	2 Dec 2003
Dr. David Goldberg/UIUC Maj. J. Vasquez/AFOSR	<i>The Design of Military Innovation: Ruggedized GAs for Robust, Rapid Solutions to Military Problems</i>	13 Jan 2004
Dr. Jae C. Oh/Syracuse University	Evolutionary Computations, Genetic Rule-based Systems, and Evolutionary Games for Real-world and Military Applications	23 June 2004
Dr. E. Wells/U. AL	Reconfigurable Hardware and Hybrid Architecture approaches for solving Evolutionary Computing Optimization Problems and performing Process Scheduling	19 July 2005
Dr. Jeff Ridder and Jason Handuber/System of System Analytics, Inc.	Mission Planning for Joint Suppression of Enemy Air Defenses Using a Genetic Algorithm	27 July 2005
Dr. A. Stoica/JPL	Evolutionary Hardware at NASA	3 Aug 2005

Our search of the literature and discussions with numerous workers in the field of EC taught us that the workhorse EC algorithm is the Genetic Algorithm (GA). We decided to continue to focus on this well studied, widely used algorithm in our in-house work. We also investigated sources of computer codes (e.g. <http://www.aic.nrl.navy.mil/galist/src/#C> , and identified gaps. We did download and make use of an experimental MatLab GA library from NCSU <http://www.ie.ncsu.edu/mirage/GAToolBox/gaot/>. Subsequently, MathWorks announced a similar product. Although we would gladly have used existing codes, we could find no downloadable examples of a distributed Island Model GA, or of a one FPGA core GA, so we developed our own. We did identify at least 4 projects that have studied or developed GA codes for FPGA, but they were either multiple chip implementations, or not written in VHDL. We are presently pursuing collaborations with two of these, one at Wright State University, and one at

the University of Alabama, Huntsville and NASA. We also determined that at this writing there is no published work demonstrating a distributed GA (or any EC algorithm) implemented in hardware on a cluster containing FPGA's at each processor node.

Labview version of GA optimizer and Ag/Ab binding bio-model ported to C

GA fitter description

A Labview version of a Genetic Algorithm based optimization tool was developed in-house at AFRL/IFTC, and was coupled with a particular bio-model to form a parameterization tool that could search for sets of parameters that fit the model to experimental data. That work was done actually done as a part of the in-house component of our involvement in the DARPA SIMBIOSYS and BIOCOMP programs. Under those programs we studied the software, algorithm, and computing requirements being encountered by program PIs, and found that one of the significantly under-addressed needs was for optimization tools, e.g. for parameterizing non-linear ODE bio-models to fit experimental data. Another motivation was that AFRL/IFTC is taking part in an AFRL level working group studying bioscience and technology and its implications on the information area, and GA methods are indeed algorithms inspired by biology.

The Labview software programming environment was used to develop the first version of the GA fitter tool and bio-model tool because it is an icon based programming language which is very efficient, mainly because programs are constructed as "Virtual Instruments", with wiring diagrams that consist of interconnected icons that implement a data flow architecture of calculations to solve the problem. No line-by-line code is written, so there are no syntax errors, making code development very efficient. The Labview language is also hierarchical.

The GA tool we wrote was inspired by generic GA concepts, e.g. as described in Practical genetic algorithms by Randy L. Haupt & Sue Ellen Haupt, New York, Wiley, c1998. Originally it was written as one integrated program, but for the purposes of this project it was broken into two independent parts, the GA optimizer part, and the bio-model part, to that the optimizer could be more easily be applied to different models. We included standard "simple GA" features such selection mechanisms based on either rank or fitness, uniform random crossover of genes during mating, low level mutations accomplished by gene bit flipping, a fitness function to grade individuals, (in the ODE case the typical least squared difference between measured and model simulations), elitism (keeping a subset of the best solutions from generation to generation), etc. Figure 1 shows a front panel shot of the Labview GA optimizer.

The GA optimizer can be used by an operator entering parameters on the front panel, or it can be called as a subroutine VI by a higher level program. A number of higher level programs were also written to characterize the performance of the GA optimizer, e.g. by averaging the results of several runs, or by running with different GA parameter settings. The primary GA parameters that can be set with controls on the front panel are described in Table 2. It also has a number of flags to control various operating modes, e.g. to easily set up to fit only some of the genes, or restrict the range of bits that are mutated. The progress of runs is monitored by graphs showing the best individual's fitness vs. generation, and a histogram of the fitness values of the current generation.

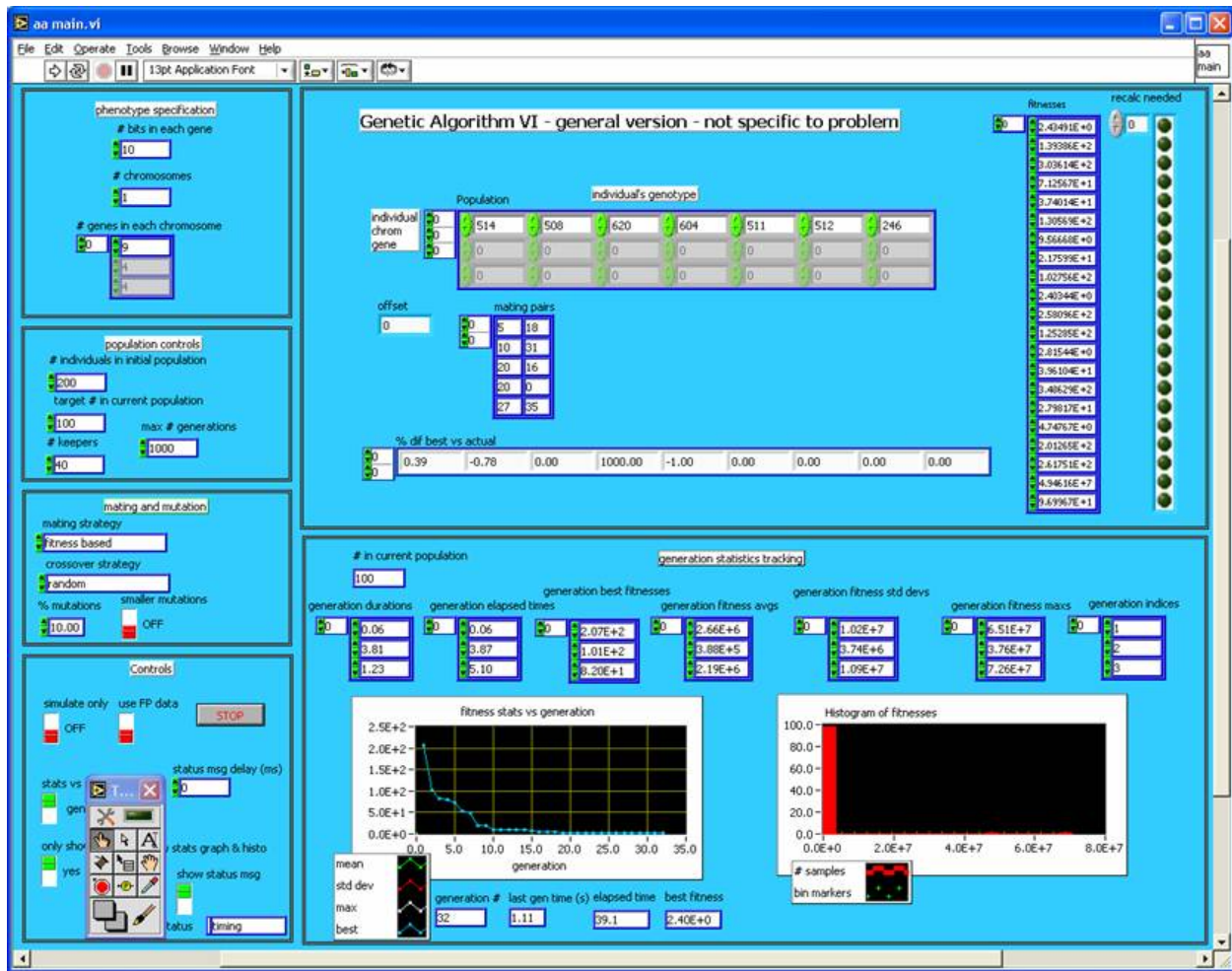


Figure 1: Front Panel of GA optimizer tool solver in Labview version.

Table 2 Primary GA optimizer parameters.

Name	Description	Typical Value
# bits in each gene	# bits in each gene	10
# chromosomes	# chromosomes	1
# genes in each chromosome	# genes in each chromosome	9
# individuals in initial population	# individuals in initial population	100-10,000
# individuals in target population	# individuals in running population	100-10,000
# keepers	# individuals used as parents to produce children for next generation	10-20% of population
max # generations	Maximum # of generations allowed	20-1,000
mating strategy	method used to rank individuals for selection for mating	fitness based
crossover strategy	method used to pass genes to next generation	random (gene by gene)
% mutations	total # bits mutated in all but best individual after mating	<10

Bio-Model Description:

The bio-model first used was based on an early version of a model being developed by Dr. Ann Rundell, a DARPA SIMBIOSYS program PI at Purdue. The description in this section was derived from the 1st quarterly status report of that contract (JON E1170068, F30602-01-2-0539).

The model shown in Figure 2 captures the fundamental kinetics of epitope-paratope bond formation between antibodies fixed to a surface and mobile Antigen encountering the surface. This model was derived from the law of mass action for the stoichiometric reaction that occurs between each antigen epitope and antibody paratope.

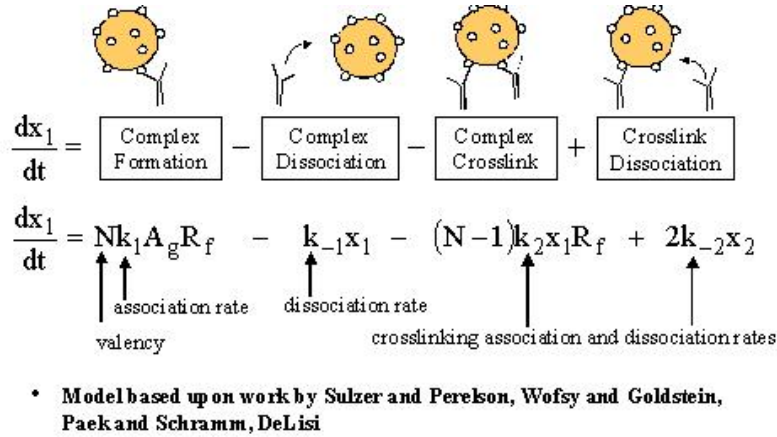


Figure 2: Structure of Epitope-Paratope Bond Formation Model

The model was programmed in such a way that the number of epitopes is variable: thus the program generates the appropriate number of differential equations and solves them. The model was simulated with reasonable parameter values that were obtained from literature. The model of antigen capture was extended to include a two compartment model of antigen binding; a two compartment model of antigen binding which explicitly incorporates effects of steric hindrances (Figure 3). Steric hindrance effects model the exclusion of surface area available for binding as Antigen binds and covers the surface.

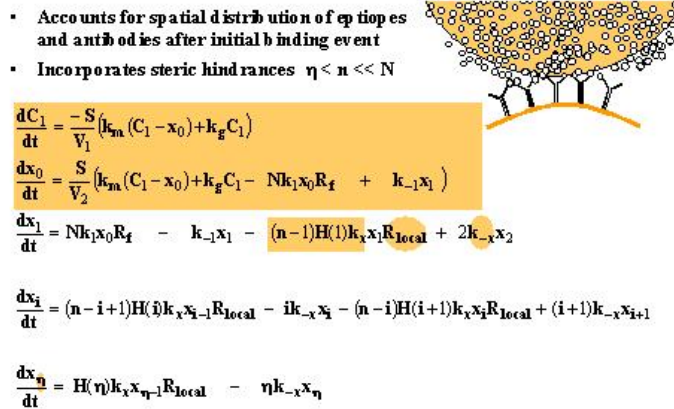


Figure 3: Two Compartment Model Incorporating Steric Hindrances

The Labview version was implemented using a couple of different ODE solvers for the equations. This first version (Lv0) made use of an ODE solver built into Labview which parsed text strings describing the equations. This proved too slow. The second one used a simple home-brew Euler ODE solver that we wrote, and it was much faster. Figure 4 shows the front panel of the equations solver. The curves on the right are binding and release concentration vs. time plots for simulated experimental data (actual) and fit or calculated data (best).

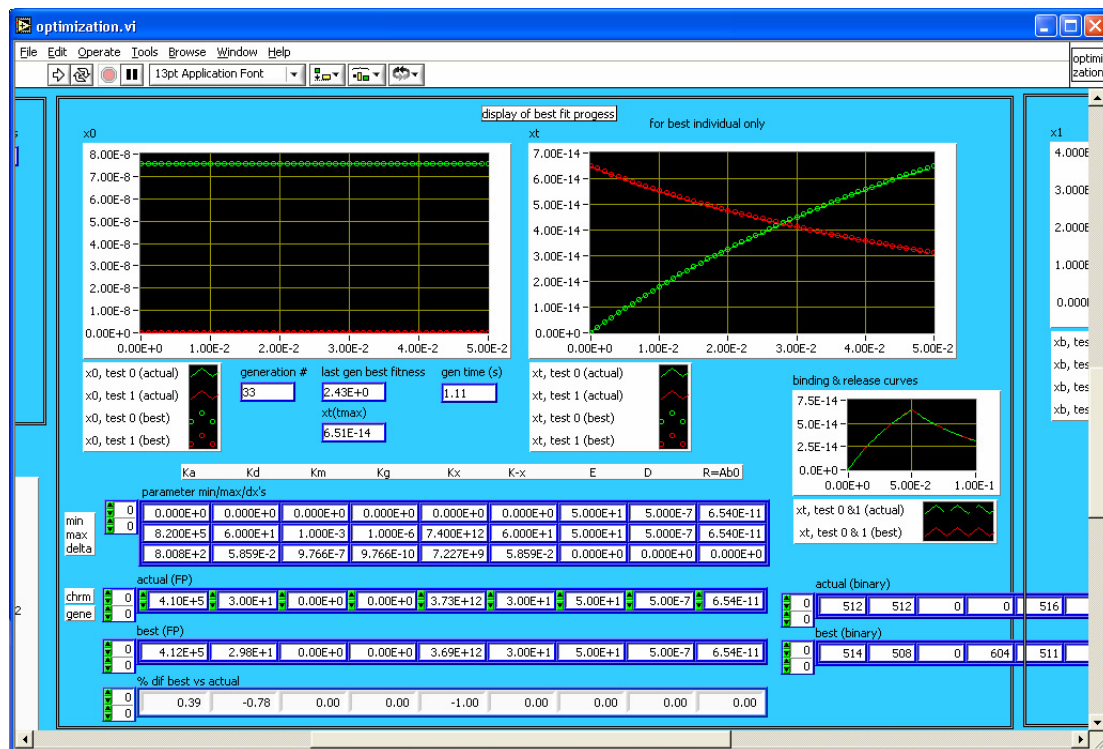


Figure 4: Front panel of equation ODE solver in Labview version.

The bio-model parameters that typically are fit or can be varied physically in an experiment are shown in Table 3. All of these parameters are assigned as genes in the GA and can be fit.

Table 3 Primary Bio-model parameters.

Name	Description	Typical Value
ka	initial association rate	4.9e+4
kd	initial dissociation rate	3.9e-2
km	transport coefficient between the two compartments due to diffusion	1.0e-4
kg	transport coefficient between the two compartments due to gravity	1.0e-8
kx	cross-linking association rate	3.7e12
kx-	cross-linked dissociation rate	3.0e1
D	antigen diameter	5.0e-7
E	antibody extension from surface	5.0e1
R=Ab0	functional antibody surface density	6.5e-11

As can be seen, these parameters have values that span many orders of magnitude. They are used as floating point numbers in the bio-model tool, but are used as binary numbers in the GA tool (typically 10 bits, to enable fitting to less than 0.1% of the searched range). Scaling is controlled by specifying a floating point min and max for each parameter. These and a number of other parameters that are used by the GA and bio-model can be input either on front panel controls or by reading a comma delimited Excel spreadsheet file.

A number of other software programs were also written to exercise the GA-model combination. One of these was a ‘model explorer’ that allows one to scan 2 parameters over selected ranges and calculate the maximum amount of binding at the end of the association phase. Another is a ‘run manager’ that runs the GA fitting process several times and gathers statistics about the results, as it is important that statements about the performance of GA’s be made in terms of statistics gathered over many runs. Other programs were run that repeated fitting runs for a chosen sets of GA parameter values, e.g. for different initial population sizes.

This Labview version (Lv1) was also ported to C (Cv1). The C version was developed using the cygwin package (a Linux-like environment for Windows, www.cygwin.com) and the GNU gcc compiler and make packages (www.gnu.org). Cv1 was written to take input at compile time from two header files, one specifying GA parameters, and one specifying bio-model parameters. It was also written so that some of the parameters could be passed in at run time as arguments on the command line, so that it would not be necessary to recompile the program when certain parameters were changed. Again several programs were written (some in Labview) to control the exercising of this C version and for gathering statistical performance results over many runs.

The performances of the Labview version (Lv1) GA–Model combination and of the port to C version (Cv1) are covered in a later section of this report.

MatLab GA and bio-models

Two MatLab versions were programmed. Both versions used the MatLab based Genetic Algorithm Optimization Toolbox (GAOT) obtained from North Carolina State University <http://www.ie.ncsu.edu/mirage/GAToolBox/gaot/> to implement the GA fitter tool. The first version (Mv1) used a bio-model ported to MatLab from Cv1 (previously ported from Lv1), including the home-brew Euler ODE solver. The second version (Mv2) used a bio-model with mode detail supplied by the PI of the Purdue SIMBIOSYS effort. Mv2 made calls to an ODE solver built into MatLab. We made modifications to GAOT to introduce the same selection, mating, and fitness evaluation functions used in the other versions.

Both of these versions worked, but they were very slow, e.g. taking on the order of a second or more to evaluate the concentration vs. time curves for a single individual. This is too slow to be of much use for parameter fitting or optimization runs that typically might require hundreds or thousands of evaluations. It is worth noting here that the Purdue PI used a method for parameterizing the model that is probably a pretty typical practice. It involved fitting a couple of parameters first, while assuming values for the others, and once those parameters were determined, fitting a couple more, etc. This approach is necessary because the methods used in MatLab's built in optimization toolbox (Fminbnd - Golden Section search and parabolic interpolation, and fminsearch - Nelder-Mead simplex search method) do not always converge when more than two parameters are fit at the same time, especially when the sets of equations are non-linear. The GA approach is more tolerant, e.g., it is able to fit four or more parameters simultaneously for these sets of equations.

It is also noted that the goodness of fit is typically calculated as the least squared difference between experimental (or simulated experimental) data and model predicted concentration vs. time curves. The bio-model predicts separate curves for singly, doubly, etc. bound Antigen, and a total bound curve is calculated by summing them. In general is easier to fit the model if many such data curves are available, but it might not actually be feasible to measure all of the curves experimentally. All of the results here used evaluation functions that included many curves, rather than just the total bound curve. If only the total bound curve can be experimentally measured, multi-stage fitting approaches as described above may be necessary. This restriction of experimental measurability may not apply in other problem domains, therefore it is still of interest to evaluate performance based on evaluation functions that use all the curves.

These MatLab versions were written to take as input a '.m' file that specified GA parameters and a '.m' that specified bio-model parameters, in a manner similar to that of the '.h' files used in the C version described above. Also, several programs were written to control exercising the tools, e.g. to run several trials and gather statistical results to characterize performance, and to repeat tests for a multiple sets of parameters. The performances of these MatLab versions (Mv1 and Mv2) are also covered in a later section of this report.

MatLab v2 bio-model ported to C

Finally, a C version (Cv2) was produced that used the GA tool and ODE solver ported from Cv1 (C from Labview) that was mentioned above, along with a bio-model ported from Mv1 (MatLab bio-model from Purdue). Table 4 summarizes the characteristics of all the various GA fitter/bio-model versions.

Table 4 Characteristics of various GA fitter/bio-model versions.
(custom means code written by IFTC)

Name	Source	GA	Bio-Model	ODE
Lv0	Labview	custom	Early custom	parsed from text
Lv1	Labview	custom	Early custom	custom
Cv1	C	custom	Early custom	custom
Cv2	C	custom	Purdue, June 02	custom
Mv1	MatLab	GAOT	Early custom	custom
Mv2	MatLab	GAOT	Purdue, June 02	In MatLab

Evaluation of speed, accuracy, convergence, and scaling performances of the Labview, MatLab, and compiled C versions

A number of tests were done to characterize and compare the performance and scaling behavior the various versions. Prior to the performance tests, some initial studies were done to identify reasonable values for the GA parameters, with the results listed as typical parameter values in Table 2. After those initial studies, performance and scaling metrics were defined to as shown in Table 5.

Table 5 Performance and Scaling metrics for GA bio-model fitter.

Category	Description
Speed	initial population solution time vs. initial population size running or target generation solution time vs. target population size average solution time for good fit
Accuracy	individual parameter fitting errors average parameter fitting error maximum parameter fitting error
Convergence	did the run terminate at the maximum number of generations?
Problem Difficulty	behavior fitting 2, 4, 6, or more parameters simultaneously
Parameter Search Range	Effect of widening parameter search ranges
Fitting Accuracy	Effect of relaxing fitting accuracy requirement
Noise and Sparse Data	Effect of fitting noisy and sparse data

Generally the results are stated as the average performances measured for a set of 20 or 30 runs of the fitting task, although only 1 run was used for some of the slower versions. Care was taken to set GA and bio-model parameters the same for each version so that very similar

calculations were being made when testing each different version (e.g. the binding and release curves had concentration vs. time values calculated for 100 points in time). The tests and the results are described in the remainder of this section.

Speed

Figure 5 shows the results of the speed Test_0 that measured the solution time of the initial population as a function of the number of individuals in the initial population. The compiled C version Cv1 was 2-3 orders of magnitude faster than the Labview version Lv1. Using the Purdue bio-model with more detail increased the execution time by less than a factor of 2 in the compiled C versions (Cv2 vs. Cv1). Moving from the all custom Labview version to the MatLab version using GAOT decreased execution times by a factor of about 3-20 (Mv1 vs. Lv1). In MatLab, moving to both the model with more detail and to the built-in ODE solver increased execution time by a factor of about 50 (Mv2 vs. Mv1).

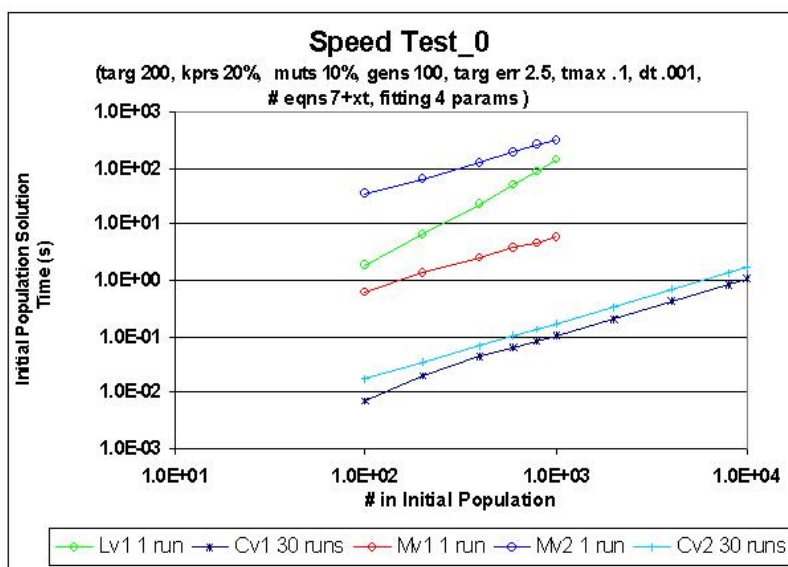


Figure 5: Speed Test_0, solution time of the initial population as a function of the number of individuals in the initial population.

The second speed test done (Test_1, Figure 6) measured the execution time for solving the first generation of the running, or target population, again as a function of the number of individuals in the target population. As expected, the curves are basically identical to Test_0.

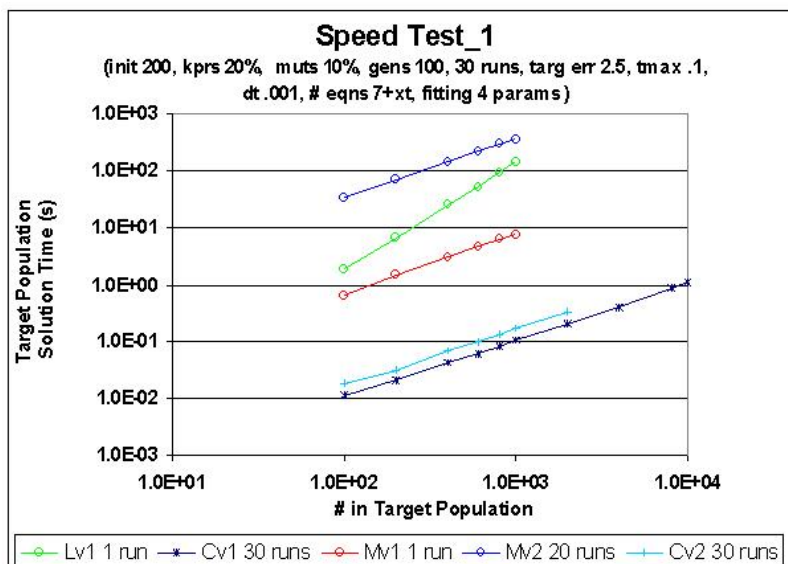


Figure 6: Speed Test_1, solution time of the target population.

Any of the fitter versions should require a similar number of executions of the ODE solver to evaluate candidate parameter sets in order to arrive at a good fit. In practice, good fitting runs may require simulating on the order of 200 generations of a population of 1000 individuals. Cv1 and Cv2 show generation execution times of about 0.1 sec for a population of 1000 individuals, so 200 generations would take about 20 sec. Mv2 shows a generation execution time of about 360 sec for a population of 1000 individuals, so 200 generations would take about 7.2e4 sec, or about 20 hours. The C versions would obviously be much more useful than the MatLab versions for doing a lot of optimization runs.

The third speed test (Test_2, Figure 7) measured the time to complete a fit. The termination criteria for this test included two conditions. One was related to the maximum number of generations, in this case 2000, and the other was related to the accuracy of the fit that we called the maximum normalized difference. The maximum normalized difference was calculated at every point on the concentration vs. time curves as $\text{abs}[(\text{best}-\text{actual})/\text{actual}]$. The fit was terminated if the maximum of these calculated values was smaller than a target value of 0.02, meaning that all points on all of the curves was fit to <2% error. We had previously used an accuracy criterion that was the sum of the normalized differences at all points, but we suspected that the sum measure could be low even with significant point-to-point fitting errors.

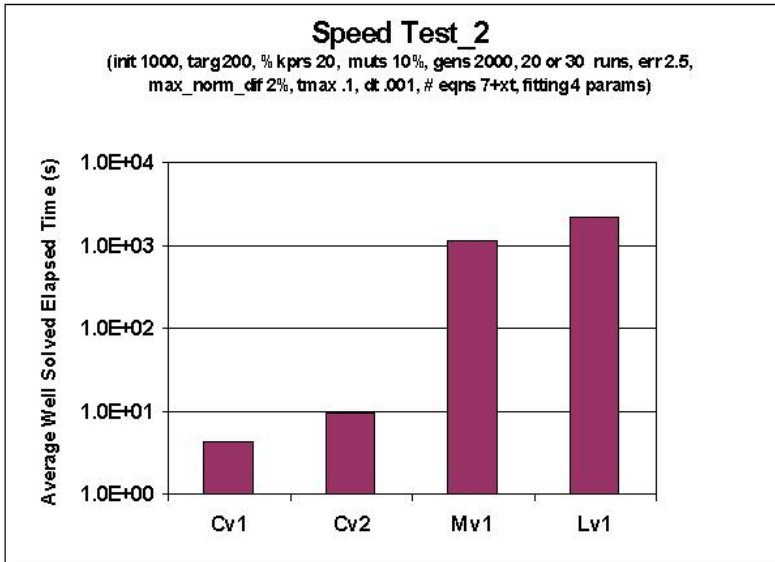


Figure 7: Elapsed time for fitting task for the versions.

Table 6 % of Runs Reaching Maximum of Generations

Version	%
Cv1	0
Cv2	0
Mv1	90
Lv1	10

An important point to note is that 2000 generations was enough for the C and Lv1 versions to do a good fit, but the Mv1 version often ran to the maximum number of generations, as shown in Table 6. We later found that the residual errors in parameter estimates were larger for this version, too. These results suggest there was a residual bug in that version, but after carefully looking at the code for our adaptation of GAOT we could not find one. We did not look for bugs in the complete GOAT code because that was beyond the scope of this project.

Accuracy

Another metric of interest is the accuracy of fitting parameters in the model. We kept track of individual parameter fitting errors, and calculated the maximum and average fitting errors after each run, and the averages of these measures over all runs. Figure 8 shows the average over 20 or 30 runs of the maximum parameter fitting error over all 4 parameters being fit. These data were measured during Test_2 described above, and we know that the MatLab GAOT version was running to the maximum number of generations, not really finishing a good fit. The results show that 4 parameters could all be fit simultaneously by the C and Labview versions to within about a percent using the 2% maximum normalized difference criterion.

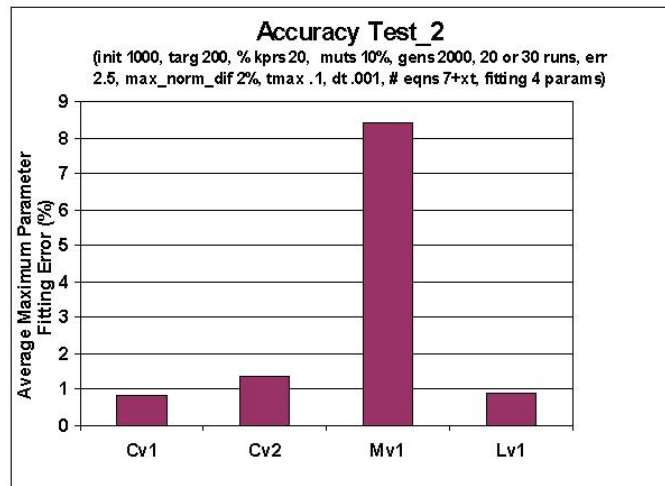


Figure 8: Average maximum parameter fitting error for the versions.

Convergence

We mentioned above that the maximum number of generations allowed for Test_2 was 2000 generations, but actually the Lv1 and C versions usually accomplished a good fit in 220-300 generations, as shown in Figure 9.

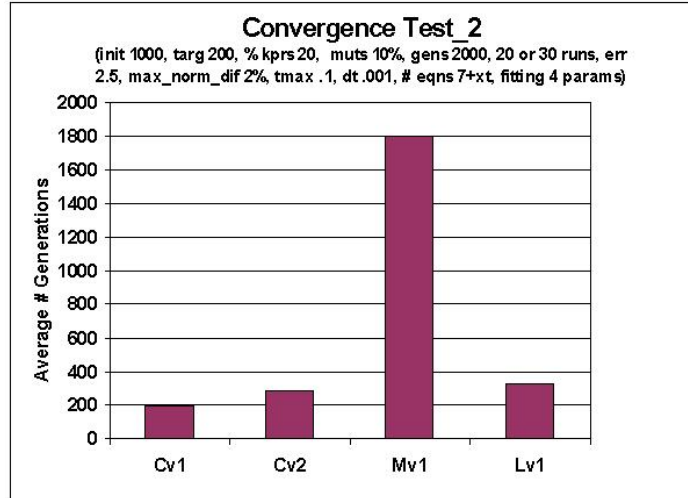


Figure 9: Average number of generations for good fit for the versions.

Problem Difficulty

Figure 10 shows the average elapsed times and average sum of parameter fitting errors for 20 runs while fitting different numbers of parameters, using version Cv1. Table 7 shows which parameters were being fit for each set of bars. Generally, fitting more parameters is harder and takes more time.

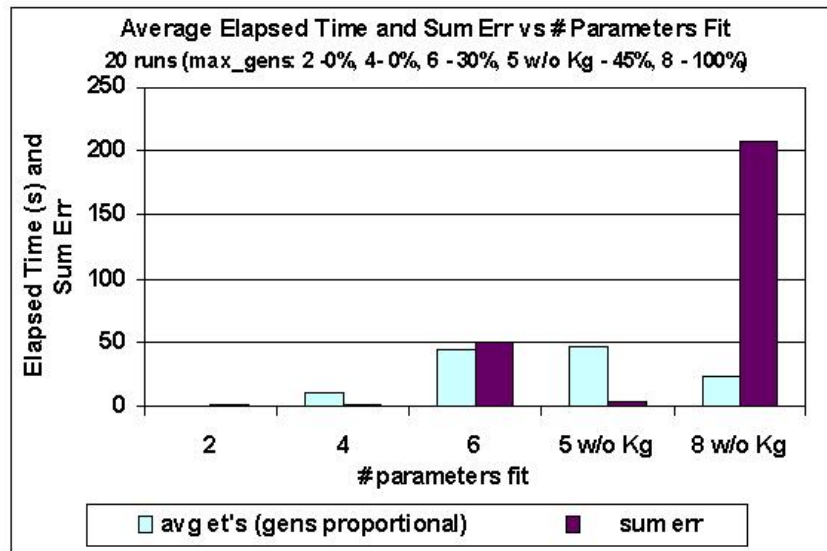


Figure 10: Elapsed time and Sum Err fitting different numbers of parameters (Cv1 version).

Table 7 Parameters being fit for sets of bars in Figure 9.

# parameters being fit	parameters being fit
2	Ka, Kd
4	Ka, Kd, Kx, Kx-
6	Ka, Kd, Kx, Kx- , Km, Kg
5 w/o Kg	Ka, Kd, Kx, Kx- , Km
8 wo/Kg	Ka, Kd, Kx, Kx- , Km, D, E, R=Ab0

The tests of Figure 10 were done with similar test conditions as the previous tests, with initial population=1000, target population=200, and the maximum number of generations was 2000 except for the set labeled 8 w/o Kg where the number in the initial population was 500 vs. 1000 and the maximum number of generations was 400 vs. 2000 because the fitting times were so long. All the runs went to 400 for the 8 w/o Kg set. The test was repeated with 2000 generations, and the results were similar (all runs went to 2000 generations).

Effect of parameter search range

Figure 11 shows the effect of parameter fitting range on performance, again using version Cv1. The label 2X means that the search range for each parameter in the model was set to be twice as wide as, and centered on the actual parameter value, e.g. if an actual parameter value was 2, the reach range would be from 0 to 4. The # generations and times go up searching larger ranges, but the accuracy of fitting parameters is still good. All of the above tests were done with a range of 2X.

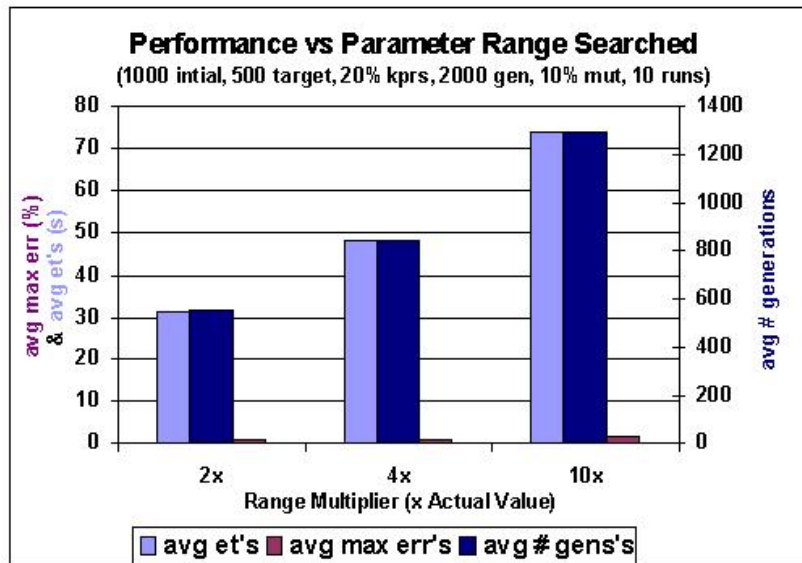


Figure 11: Effect of parameter search range on performance.

Fitting with relaxed accuracy requirements

Figure 12 shows the effect of relaxing the goodness of fit requirement, i.e. increasing the maximum normalized difference termination criteria target.

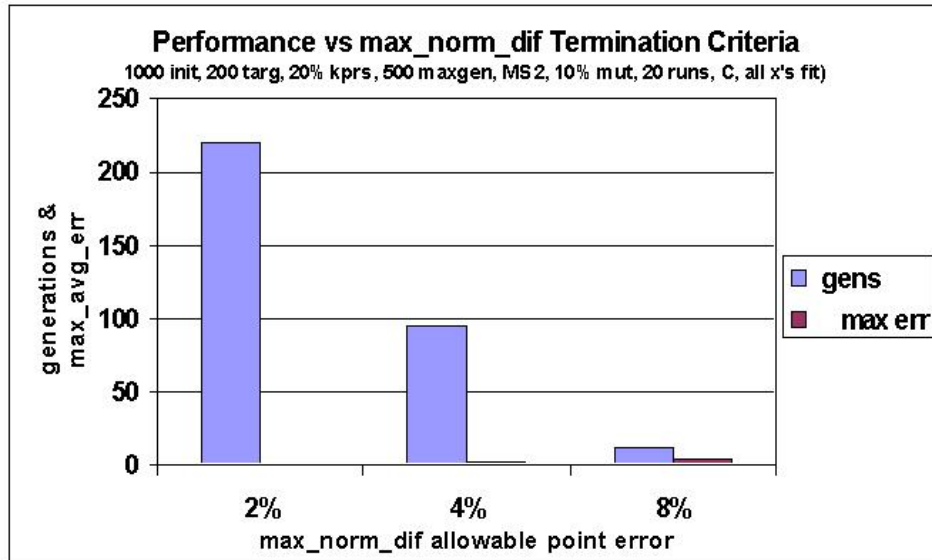


Figure 12: Effect of fitting with relaxed accuracy requirement.

Fitting noisy and sparse data

Some testing was done at the suggestion of the Purdue PI that looked at fitting noisy and sparse data. These simulations introduced amplitude noise by adding a random 0-20% noise value to each of the data points to be fit. The tests with sparse data simply used a subset of the data points normally fit, with every nth point selected for the set to be fit. Figure 13a. shows a fit done with no noise, and all data points for reference. Figure 13b. shows a fit done with noisy data (maximum 20% noise). Figure 13c. shows a fit with only every 10th data point used for the fit. Figure 13d. shows a fit done with a max of 20% noise and using every 10th data point. These tests were done with only 20 generation runs, so the fits of noisy data would probably have been closer to a least squares fit of the data used if more generations had been used. These tests were done with MatLab version Mv1.

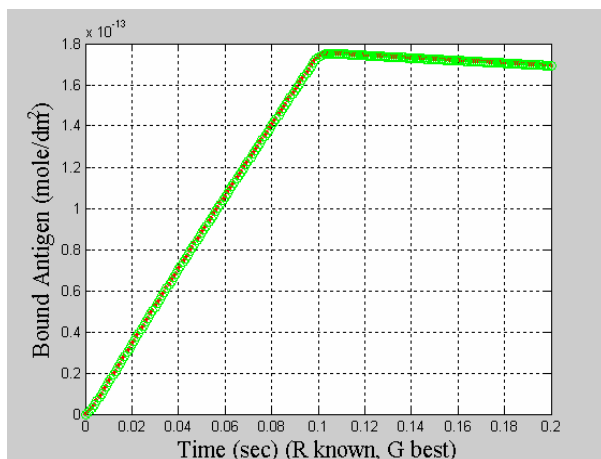


Figure 13a. Fitting all data points, no noise.

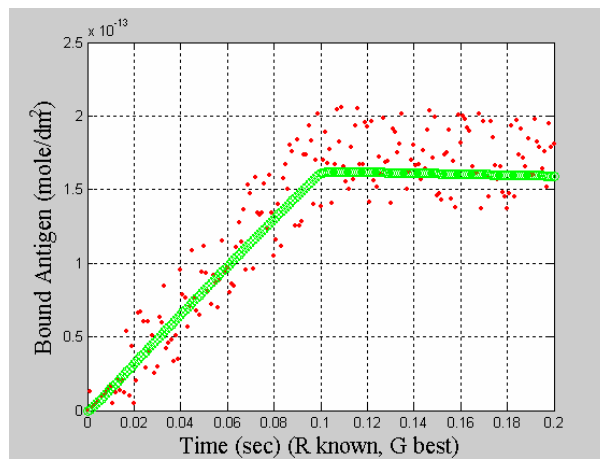


Figure 13b. Fitting all data points, 20% random noise.

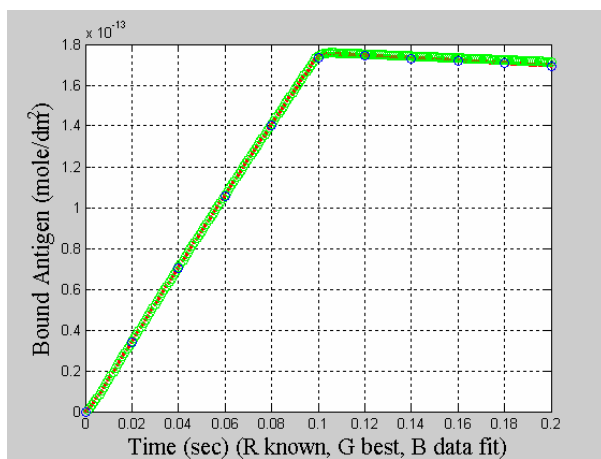


Figure 13c. Fitting every 10th data point, no noise.

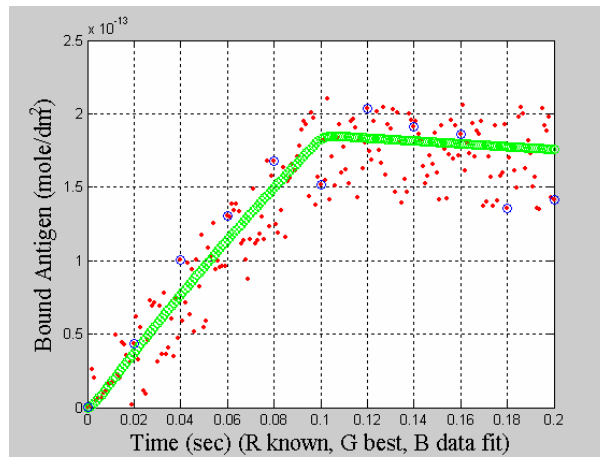


Figure 13d. Fitting every 10th data point, 20% random noise.

Evaluation of GEPASI Bio-model Simulator

While attending the summer 2003 DARPA SIMBIOSYS PI meeting I became aware of a software package called GEPASI that was developed by [The Mendes group](#) at the [Virginia Bioinformatics Institute](#) located in the Corporate Research Center at Virginia Tech in Blacksburg, VA. From web site <http://www.gepasi.org/> "...GEPASI is a Microsoft Windows program intended for the simulation of the kinetics of systems of chemical and biochemical reactions. The program is aimed at the study and teaching of the behavior of such systems. GEPASI is able to simulate the steady-state and time-course behavior of reactions in several compartments of different volumes. The user supplies the program with information about the stoichiometric structure of the pathway, kinetics of each reaction, volumes of the compartments and initial concentration of all chemical species. The program then builds the differential equations that govern the behavior of the system and solves them. Results are produced in a flexible way so that data can be imported into spreadsheets or other data processing programs. The data can also be plotted in 2D and 3D graphs directly from the program (by using the package gnuplot that is distributed with GEPASI). GEPASI has the ability of scanning ranges of values of the system parameters and produce a mapping of the behavior of the system within these ranges. ..."

GEPASI also can be used to fit model parameters, and can do so with a number of classical methods, as well as an "Evolutionary" method. I downloaded GEPASI and worked with it for about a week. I set up a simple version of the equations for the Ag/Ab binding system, and tried out some of the parameter fitting mechanisms. I did not learn how to input the model exactly using the set of canned reactions they provided, so I implemented it using a sum of custom reactions they allow the user to define. I verified that it reproduced the binding curve pretty closely. I experimented with fitting the k_a parameter. k_a appears in more than one term, and I tried fitting its appearance in only one of the terms. I found that only the "Evolutionary" method worked. The others failed to converge. The others included Hook and Jeeves, Levenberg-Marquardt, Levenberg-Marquardt multistart, Nelder and Mead (simplex), and Simulated Annealing. There was a method called Random, and it did better than the others, although much poorer than the Evolutionary. Each of the methods had a set of parameters to set up, and I experimented with them all but could not find sets that would make the classical methods work. I suspect the reason is that the equations are non-linear in k_a .

I'm not sure if it is possible to set up the set of equations in GEPASI exactly as they are in the Purdue model, or whether all the occurrences of each parameter can be linked to each other and still be fit.

Java/OAA wrapped PC GA ODE Parameterizer version prepared as BioSpice agent.

A version of the PC GA ODE parameterizer was developed and tested that met the requirements for contribution as a functional agent operating under the BioSpice Open Agent Architecture (OAA 2.3.0) software environment. Following the pattern of certain other BioSpice agent contributions, this version wrapped a C executable in OAA Java. It included an Apache Ant Java based build tool to control compilation on destination machines, as well as test case data and results. We provided this version to the IF focal point for the SIMBIOSYS program.

Web Browser Interface PC GA ODE Parameterizer Version

During the winter semester break of 2004, a unique interface was developed that explored ways to enable remote, non-programmers who might want to experiment with such tools to easily control runs with different values of various parameters in the GA and Model. This type of interaction is often necessary in order to discover sets of parameters that solve the problem well, or to test performance while running a Model with different settings or under different conditions. The idea is to make a user interface that is independent of the specific GA or Model C codes, i.e. that would work with other Models contributed by other workers and integrated with the GA ODE parameterizer, and that would not require the user to be a programmer in either the source programming language of the GA or the Model executables. The basic approach is to present the user with at a web page that allows the specification of a set of variables, and values for them, and that includes functionality that automates the process of updating and running the GA and Model codes. Figure 13 shows an outline of the parts of the web interface.

The first part is a web page with a 3 column list that allows the user to specify a one or more Model variables and their specific values to be used in a run. The first column is for the names that the biologist prefers to use for each parameter when working with the Model. The second column is for the names the Model code programmer used for the variables in the Model code, and the third column is for the values the user wants to apply to these variables for the run. The user then clicks the Submit button on the web page, and three things happen. First, a file is written to disk that contains the parameter mapping names and values. Second, another web page pops up and reports progress to the user during the ensuing re-compilation and run process. Third, it causes a batch program to run that looks at the information in the disk file and runs a C application called `Create_page.exe` that in turn produces the source code for a second web page (`GA_params.html`).

The second web page is a reusable interface to the GA Parameterizer. It pops up and allows the user now to enter GA parameters and execute runs of the fitting procedure, e.g. with different population size, number of generations, etc. After entering GA parameter values, the user hits the submit button which triggers saving a GA parameter information file to a disk file, a progress update on the reporting page, and the launching of another batch file called `GA_v2web.bat`. This batch file runs a C application that parses the saved Model and GA parameter data files, produces a Systems Biology Markup Language (SBML) file that specifies all the desired GA and Model variable values for a run, and then runs the GA/Model executable. The GA/Model executable in turn reads the SBML file with GA and Model variable data when it starts, and applies the variable values before running. This sequence of steps worked well, and it accomplished the goal of enabling a complete non-programmer to interface efficiently with the fast, compiled C codes for the GA and Model.

An important feature and advantage of this approach is that it enables a remote, non-programmer user to use the GA/Model ODE Parameterization tools using only a web browser interface, which is a free, open-source, non-platform specific mechanism.

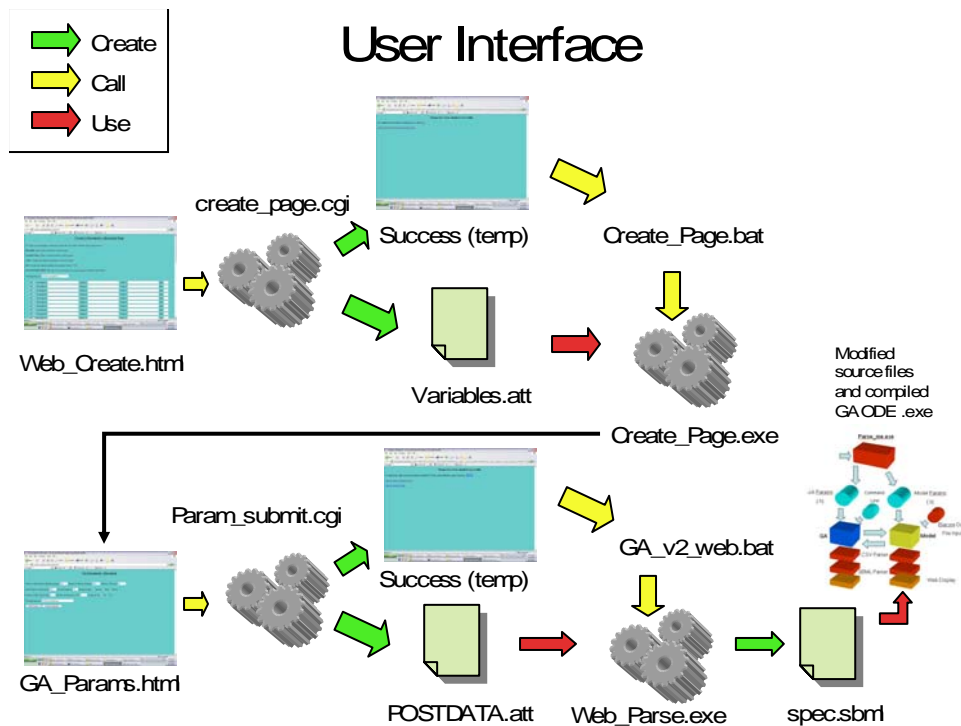


Figure 14: Outline of web browser interface for GA ODE Parameterizer.

Summary of work in Spiral 1 (PC platform)

We found that a GA based optimizer could solve the Non-Linear ODE Parameterization problem for our test case problem, and that it was superior to common methods currently being used by some DARPA program PI's because it worked on full, unreduced, non-linear models, and can easily deal with sparse and noisy data of the sort provided by real world experiments, and in our case it operated much faster as well. We also developed two versions of the tool with enhanced user interfaces. One is a BioSpice agent version, and one is a web browser interface version.

The work in this spiral was presented to the 2003 Scientific Advisor Board during their visit to AFRL/IF, in the Advanced Computer Architecture focus area poster session. It was also discussed with relevant DARPA SIMBIOSYS and BIOCOMP program PI's at various PI meetings over the last 2 years. Finally, it was reported at GECCO 2004, in Seattle (see Appendix A).

Spiral 2: (Cluster platform)

GA ODE Parameterization Problem

During the summer of 2004 we developed and evaluated two parallel versions of the GA ODE parameterization tool. One was based on a "Farming Model" GA, and one was based on the Island Model" GA. These are so called "multi-deme" GA's that divide up the work of evaluating the fitness of individuals in the population across multiple processors in a cluster. In

the farming model, one processor maintains the large population and passes groups of individuals to all other processors for evaluation, and waits for them return before starting the next generation. In the Island Model, a group of processors is connected in a ring topology, and each processor breeds a separate, smaller population, but periodically passes a few of its best individuals to another processor, and receives a few from another processor. The Island Model GA has better performance as processors are added to the group because it requires significantly less total communication time.

These parallelized codes are written in C, and use the Message Passing Interface (MPI) protocol for communication among processors. This work was mostly done in the summer of 2004 by Kevin May, a Summer Engineering Aide, assisted by Dr. Larry Merkle, Rose Hulman Institute of Technology, a Visiting Summer Faculty Research Program participant.

These codes were instrumented with time profiling tools in order to analyze the amount of time the processors spend computing and communicating. This information is used to help manually optimize the communications among processors. Figure 15 shows a typical time profile for the Farming Model GA after optimizing. As a generation runs on the master processor, it is busy during the GA sort, mate and mutate (top row, orange), but the other processors are idle (other rows, light blue) as they wait for work. This is followed by rapid transfers of individuals from processor 0 to the others (row 0, white and other rows green). Then the master processor is idle as it waits (top row, green) while the other processors process their individuals (other rows purple). Then the other processors return their calculated fitness values (other rows, white) and the master processor receives these values (top row, white) and begins the next generation.

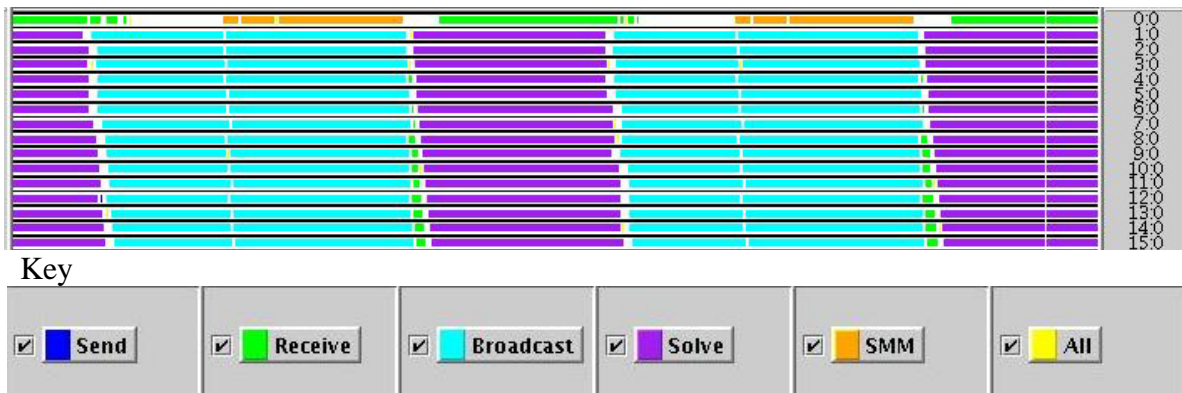


Figure 15: Jumpshot time profiling of Farming Model distributed GA (26 processors, beyond speed-up plateau).

Each row represents a processor (top row is the master processor), and colors indicate compute and communicate times. Master processor is only busy during top row white and orange, and other processors are busy only during purple. Large amounts of green and light blue times are spent waiting for communications.

A standard figure of merit for assessing the scaling performance of distributed applications is the speed-up curve, which measures how the performance of the application changes as more processors are used in the group. Figure 16 shows a speed-up curve for our best tuning of the Farming Model distributed GA, along with theoretically ideal performance. Performance plateaus at about 10 processors.

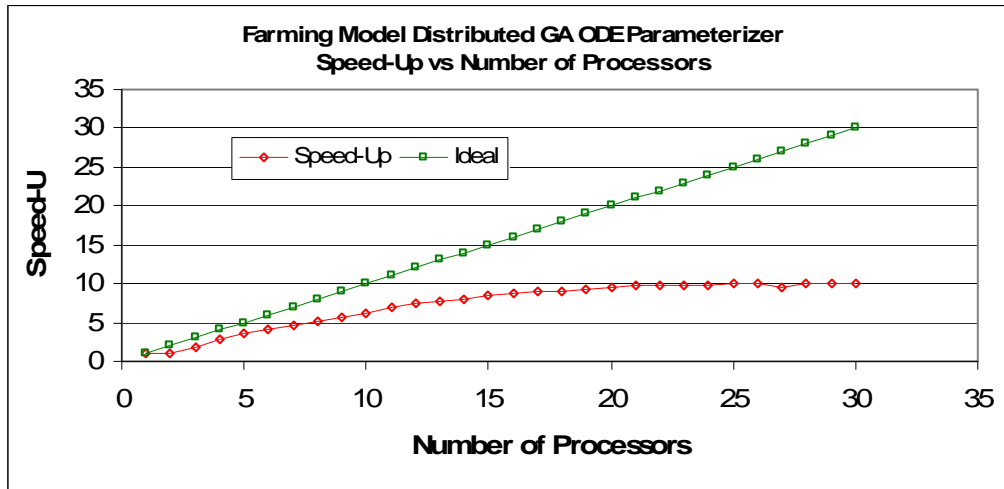


Figure 16: Speed-up curve for best tuning of the Farming Model distributed GA.

Ideal performance is rising line of slope 1. Performance does rise as up to about 10 processors are added, but plateaus.

Island Model distributed GA

An Island model distributed GA was implemented that cures some of the inefficiencies of the Farming Model GA. This model uses a group of processors connected in a ring topology. Each processor breeds a separate population, or “deme” of individuals. However, after every epoch of a certain number of generations, each processor passes a few of its most fit individuals to the processor after it in the ring, and receives a few from the processor before it in the ring. Each processor runs an identical program, although the 0th processor is the master and does certain bookkeeping tasks. Communication between the processors takes place only at the end of epochs, and when a solution has been found and the run terminates. Time profiling and communication tuning was also done on this code using Jumpshot. Figure 17 shows a speed-up curve for the best tuning. The speed-up is not perfect, but quite good, with no plateau up to about 29 processors. The curve is jagged because the points are the average of only 3-5 runs.

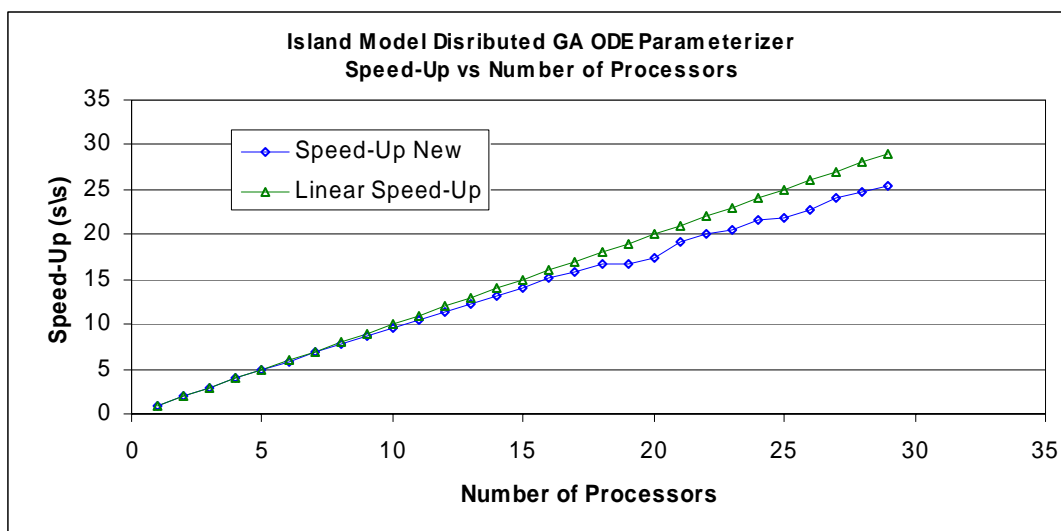


Figure 17: Speed-up curve for distributed Island Model GA ODE Parameterizer.

DNA Code Word Library Generation Problem

During 2004 we also developed both single PC and distributed Island Model parallel versions of a second application of a GA optimizer. This second application involved the DNA Code Word Library Generation Problem, which is of current interest to workers in AFRL/IFTC and elsewhere. This problem involves composing highly constrained sets of pairs of short DNA oligo-nucleotide strands, e.g. about 16 base pairs long. The pairs in the set must each consist of two strands that are perfect reverse complement halves that bind well to each other, but not well to any other strand in any of the other pairs in the library. This is a hard problem that is known to be NP-complete, and at least 4 University groups are actively working on it. There are important applications of such Code Word Libraries in the design of bio-assay micro-array chips, self-assembly of nano-structures, and in schemes for data storage and computation using bio-molecules. Random search and exhaustive search have proven ineffective for building large libraries, and current techniques use stochastic and heuristic methods.

Our approach to this problem starts building a library by finding one pair using random search. It then breeds additional words using a GA guided by a multi-objective fitness function that measures the string edit distance (calculated by the Levenstein Martix) and that also counts the number of pairs presently in the library that reject a given candidate pair. The GA uses an efficient mutation heuristic that chooses a base pair to mutate at random, checks the fitness of words with all possible single base changes at that position, and uses the mutation that improves fitness the most.

Our distributed Island Model GA was modified to provide communication among processors for 3 cases. The first case involves passing a few fit individuals around the processor ring at the end of each epoch of generations. The second case occurs when any processor finds a new code word pair that can enter the library. This is an infrequent occurrence, but the new word pair must be passed to all processors. The third case involves terminating a run when any of the processors has found the desired number of pairs, or exceeded time or generation limits.

Again the code was instrumented with time profiling tools and the communication modes and placements in the program sequence were tuned. As for any stochastic optimization method that uses random number, results must be analyzed in terms of the average performance of many runs. We developed Labview run manager and data analysis tools to automate this work. Basically, the run manager is a GUI that enables the user to specify run time switches for the GA/DNA Code application that control the run, e.g. the number of DNA base pairs in the code, the number of code words desired, the GA parameters, number of processor nodes to be used, etc. It outputs a text file that is a script that is used to invoke a set of runs on the cluster platform. Each run on the cluster produces a text file that has information about the words found, when they were found, their fitnesses, etc. The analysis tool reads these files and presents plots of words found vs. time, and of the speed-up curve. A typical experiment might consist of finding 100 pairs of words using 1 processor, repeating the run 30 times, and then doing this for 2, 3, ..., 30 processors, resulting in 900 data files. Figure 18 shows an example of the front panel display of the data analysis tool for the case of 5 runs of finding 20 words, repeated using 1-30 processors. The plot on the left shows the progress of each run, and the plot on the right shows the speed-up here are three curves in the speed-up plot. The red curve shows the theoretical ideal linear speed-up. The blue curve shows a speed-up curve where each point is the average of all runs at the corresponding number of processors that actually found the desired number of words (some may not complete within the chosen maximum time or number of generations). The green speed-up curve is for censored data that has excluded runs that did not find the desired number of words or for which the run time is outside one standard deviation of the average run time for that number of processors.

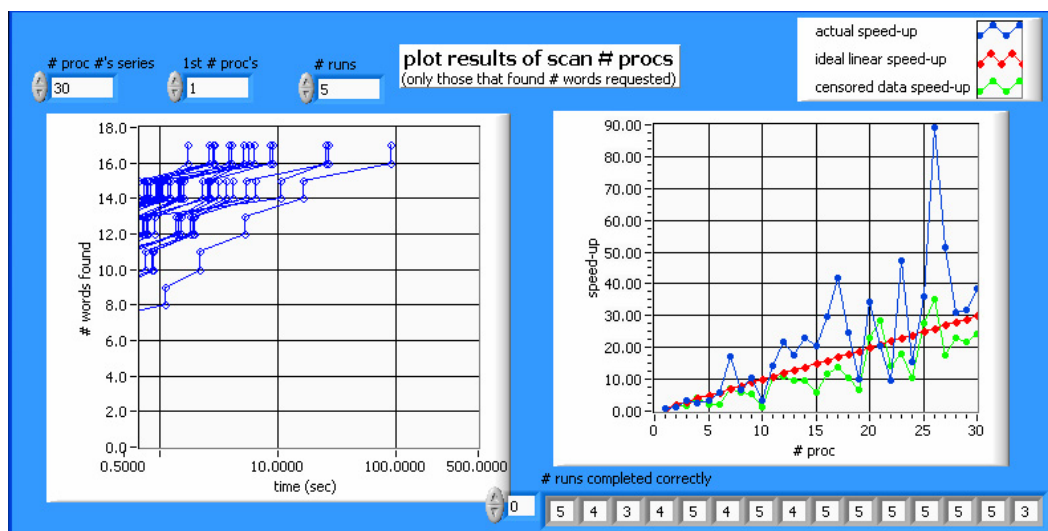


Figure 18: Multiple run data analysis tool front panel display.

Data shown indicates near linear speed-up.

We attended a 3 day conference in March 2005 at SUNY Geneseo <http://www.geneseo.edu/~macula/DNAWordConf> that brought together the top workers on this problem from 4 Universities to discuss approaches and exchange results. This led to further collaborations among several of the attendees which have since shown that the Markov probability guided Library Generation method of Dr. Tony Macula/Geneseo the fastest and most

efficient in the world. We compared the performance of that method with the distributed Island Model GA/DNA Code Word Library application, for a particular sub-case of the problem, i.e. using only the Levenstein matrix measure for constraint checking for stem size 1.

Figure 19 shows the results of this comparison. The curves show the average number of words found vs. time for multiple runs of the algorithm, and for the cases of 1 and 16 processors. The GA actually finds words faster up to the time at which words become very difficult to find. We did, however, observe that the Markov method did continue to find a few more words after that point, but we have not determined why at this writing. It may be because the criteria used for choosing words to add to the library early in the process is different for the two algorithms, and it might be that the GA would find as many words if it was run longer. At any rate, this work has led us to consider an improved heuristic for guiding mutations in the GA method that will be useful in harder versions of this problem, i.e. ones that consider stem sizes other than one. A stem is a string of adjacent matching bases. The mutation locates and mutates base positions in words that hurt fitness by extending a stem, rather than mutating randomly choosing parts of the words. We have coded this heuristic, but not incorporated it at this point.

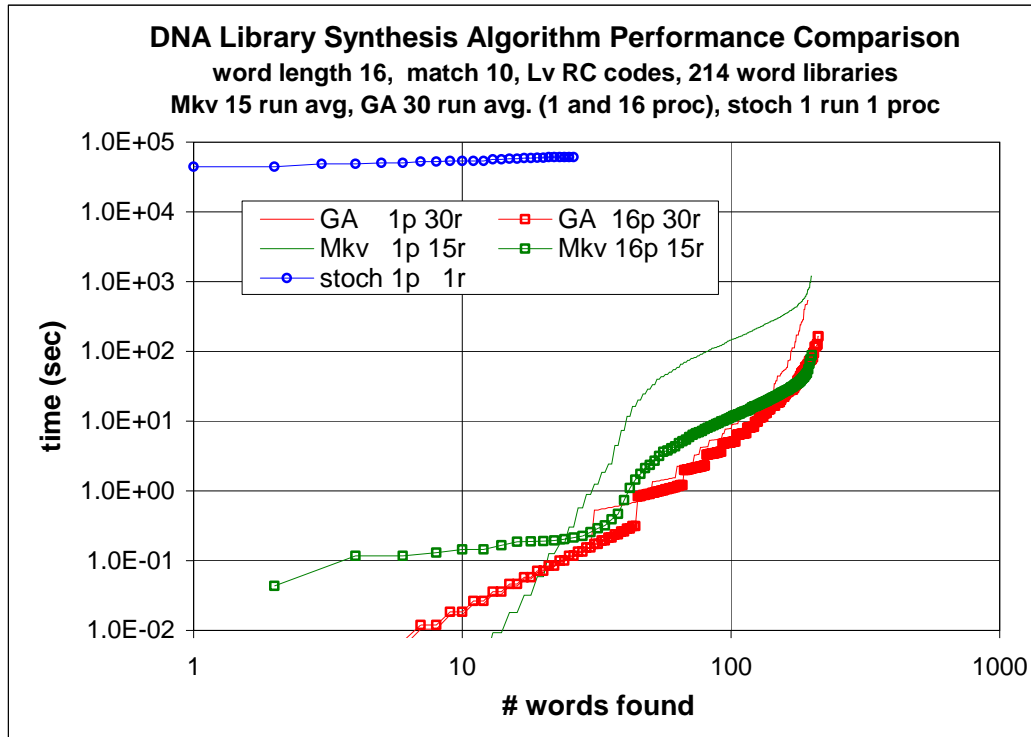


Figure 19: Comparison of Markov, GA, and stochastic DNA Code Word Library Generation methods.

GA (red) finds words faster than Markov for both 1 and 16 processor cases. Markov (green) found more words for 1 processor case. Stochastic (blue) is very slow.

Application of distributed GA to Sensor Network Energy Management:

During the summer of 2005, a Visiting Summer Faculty Research Program participant, Dr. Qinru Qiu/Binghamton University, Binghamton, NY, applied the distributed GA code we developed under this project to a third problem. That problem is the Distributed Sensor Networks Energy Management Problem, a multi-objective optimization problem. This problem involves determining best assignments of computation and communication tasks to nodes in a network, with competing objectives of providing good network service and conserving battery resources. She was able to demonstrate that the distributed GA can readily solve this problem for a test case involving a particular version of a set of network tasks, task costs. She studied a unique approach that assessed using the distributed sensor nodes themselves to solve the Policy optimization problem using a distributed GA. Typically this hard problem is pre-solved before deployment off-line, or possibly in-network by communicating with a centralized off-network node, which may have high costs in terms of communication, energy use, and result in non-stealthy exposure. She was one of 5 of 20 applicants awarded a small follow-on effort to continue their summer work, and will augment the model and tasks, and investigate GA tuning to ensure the ability to compute robust solutions.

Summary of work in Spiral 2 (Cluster platform)

A distributed Island Model GA optimization solver was developed and successfully applied to three problems, Non-Linear ODE Parameterization, DNA Code Word Library Generation, and Sensor Network Energy Management. This distributed version exhibited good speed-up scaling vs. number of processors on a 30 node cluster. The DNA Code Word solver exhibited performance that we believe rivals the best know algorithms in the world. The work on this spiral was reported at FNANO 2005 (see Appendix B), and at GECCO 2005 (see Appendix C).

Spiral 3 (Hardware Accelerated Platforms)

The third spiral addresses the ultimate path to extreme algorithm speed-up, i.e. hardware acceleration. Since integer problems as much more amenable to hardware speed-up, we will work on the GA/DNA Code Word Library Generation problem in this spiral, not the ODE Parameterization problem. We are taking two approaches to hardware acceleration. The first is uses a Higher Order Language (HOL) translation tool to produce a VHDL version more or less directly from the C version. VHDL is the language required for our tools that synthesize a personalization of a hardware FPGA. Actually, to use these tools, it is necessary to augment the C version code with data streams constructs first. The second approach is to manually rewrite the C routines in VHDL. Both of these approaches require a mind set change by the programmer, i.e. from the sequential line by line code execution model of C, to the multiple concurrent process execution model of VHDL and hardware. This does bear some resemblance to moving from sequential C on one PC to a distributed cluster running many copies of the concurrently, but in the VHDL model the parallelism extends to the lowest level of the code. We will start by addressing a single PC, single FPGA chip version of the DNA Code Word Library Generation tool, with a future goal of developing a distributed version under this spiral.

HOL tool version

We researched many of the HOL tools being marketed today, and chose one to evaluate called Co_Developer from Impulse, Inc. It requires the user to augment the original C source code for an application with constructs that fashion a data path with queued streams passing data between blocks of combinational logic and registers implementing calculations. We obtained temporary trial licenses from Impulse over the course of about a year, and eventually did produce a version that simulates correctly at the level of C in the Co_Developer Application Monitor. As far as going the next step to VHDL is concerned, our application is a moderately complex one, and we basically acted as a Beta site, testing current revisions of the tool as bugs were fixed. However, to date we have not been able to successfully compile to a VHDL version that will load and execute in ModelSim. At this point progress is still being made, and we would like to continue working at a low level of effort with the Co_Developer team at Impulse, especially on the GA algorithm part of the problem. There is still hope that this tool path may help us avoid a lot of work rewriting the GA in VHDL. At the least it would give us HOL tool version size and speed data points to compare against a handcrafted version, and this is valuable to us.

One shortcoming of the HOL tool path is that we don't think that the tool is capable of unrolling a doubly nested loop containing a calculations into a two dimensional array of hardware cells (e.g. a systolic array). As human designers we can envision this approach, but we are not sure that software tools exist to execute this type of design automatically, at least in the Impulse tool. There may be systolic array generation tools available for this somewhere.

Hand crafted VHDL version

Speed profiling of the C code version with a GNU application called gprof <http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html> showed that the Levenstein matrix calculation consumed over 98% of the time, as shown in Table 8, so this routine was selected for implementation in hardware first. During the summer of 2005 we developed two versions of synthesizable VHDL Levenstein matrix hardware accelerators, a ripple through version, and a time multiplexed systolic array version.

Table 8 Time profiling study of GA/DNA Code Word Library Generation application showing time consumed by subroutine (produced with GNU gprof).

%	Cum	self	self	total		
Time	sec	sec	calls	s/call	s/call	name
98.13	46.77	46.77	13115396	0.00	0.00	do_matrix_v6
0.65	47.08	0.31	6603415	0.00	0.00	i2s
0.44	47.29	0.21	90836	0.00	0.00	do_checks
0.38	47.47	0.18	3	0.06	0.07	clean_up_pop
0.23	47.58	0.11	272685	0.00	0.00	s2i
0.08	47.62	0.04	90895	0.00	0.00	compliment_x_str
0.06	47.65	0.03	91835	0.00	0.00	are_you_in_there
0.02	47.66	0.01	93753	0.00	0.00	pop_to_word
0.00	47.66	0.00	90895	0.00	0.00	compliment_reverse_x_str
0.00	47.66	0.00	960	0.00	0.02	smart_flip_2
0.00	47.66	0.00	63	0.00	0.50	find_fitnesses

The tool path we used includes the ModelSim Xilinx Edition II design environment from Mentor Graphics, the ISE 6.1i Service Pack3 synthesis tool from Xilinx, and we targeted the FPGA chip in our cluster with hardware, the XCV6000-4FF1517C from Xilinx.

Figure 20 shows the basic Levenstein Matrix calculation that is the main work of the do_matrix_v6 subroutine that dominates the calculation time. Word strands are presented along the top and left edges of the matrix. Each cell along the top row (and left column) is initialized to 0 and set to 1 if the bases at the top and left edges aligned with that cell are the same, or if the cell to the left (or top) is already a 1. In the inner cells, the cells are initialized to 0, and are set to the maximum of 3 values, the value of the upper, left, and upper left adjacent cells, except that 1 is added to the upper left adjacent cell value if the edge bases aligned to the cell are the same.

Levenstein Matrix

	A	G	T	T	C	A	G	G	A	C
T	0	0	1	1	1	1	1	1	1	1
G	0	1	1	1	1	1	2	2	2	2
A	1	1	1	1	1	2	2	2	3	3
C										
C										

Figure 20: Levenstein Matrix Calculation

Only the systolic array version of this array will be discussed here. The ripple through version was a good first step, but was slow (10MHz). Figure 21 shows the array of cells along with register arrays along the top and left edges that sequence word pairs into the edge cells. It also shows a breakout of one cell in the format of an entity defined in VHDL. The U, L, and UL inputs are simple signals carried in on wires, but the A and B inputs are signals latched into registers in the cell. The ans output of the 4_Bit_Compare sub-cell is also latched in the cell by a register. Actually, the output of the A and B registers are outputs as well, as they connect to the A and B inputs of adjacent cells below and to the right.

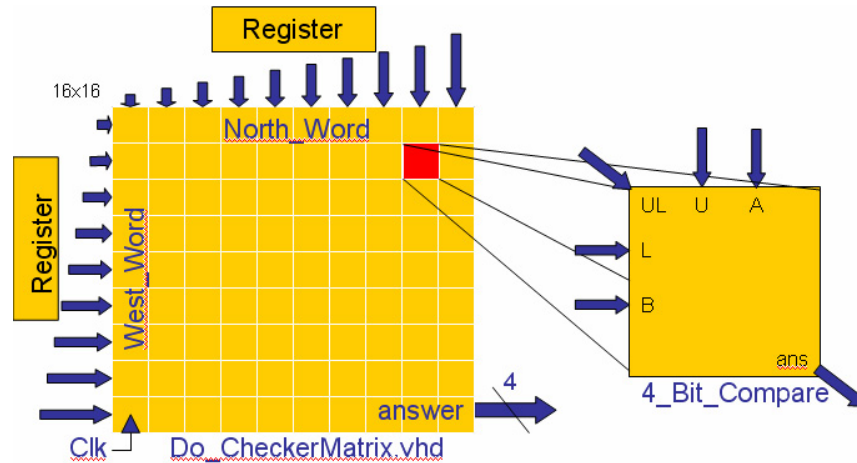


Figure 21: Levenstein matrix calculation array implementation.

The present design is a time multiplexed systolic array that calculates comparisons for 16 word pairs simultaneously that flow along diagonal lines down through the array from upper left to lower right. The shift registers at the edges of the array delay the presentation of the upper bases of the words to the edge cells as needed to synchronize with the wave of calculations for the word pairs. Inside the array, the base pair tokens are shifted down columns and across rows. Also, the matrix is operated in a checkerboard fashion, with one half of the cells at any clock loading inputs and with the other half of the cells calculating outputs. On the following clock, the opposite happens. The “latency” of the array is 16 words, i.e. the answer for the first word pair flow out the bottom right corner after the 16th word pair is processed. After that, answers flow out of the lower right cell every two clock cycles. A more complete description will appear elsewhere.

Figure 22 shows a higher level functional block diagram that includes entities for on-chip SelectBlock memories to hold the GA population, the fitness values of the population, and the words of the DNA Library. It also shows a hardware entity called MemBlock.vhd that sequences the fitness evaluation of all the individuals in the population against all the words in the library, and stores the results. Another entity is also shown that is a simple software test bench that takes the place of a GA for testing purposes at this point. The design of Figure 21 has been synthesized, and software simulations show that it operates correctly.

We have submitted two papers that have been accepted on the work in spiral 3, one to the STAR2005 meeting in Dayton, OH in Aug., and one to the 8th annual NASA Military and Aerospace Programmable Logic Devices (MAPLD) International Conference, Washington, D.C., Sep 05 <http://klabs.org/mapld05/admin/cfp.html> ,

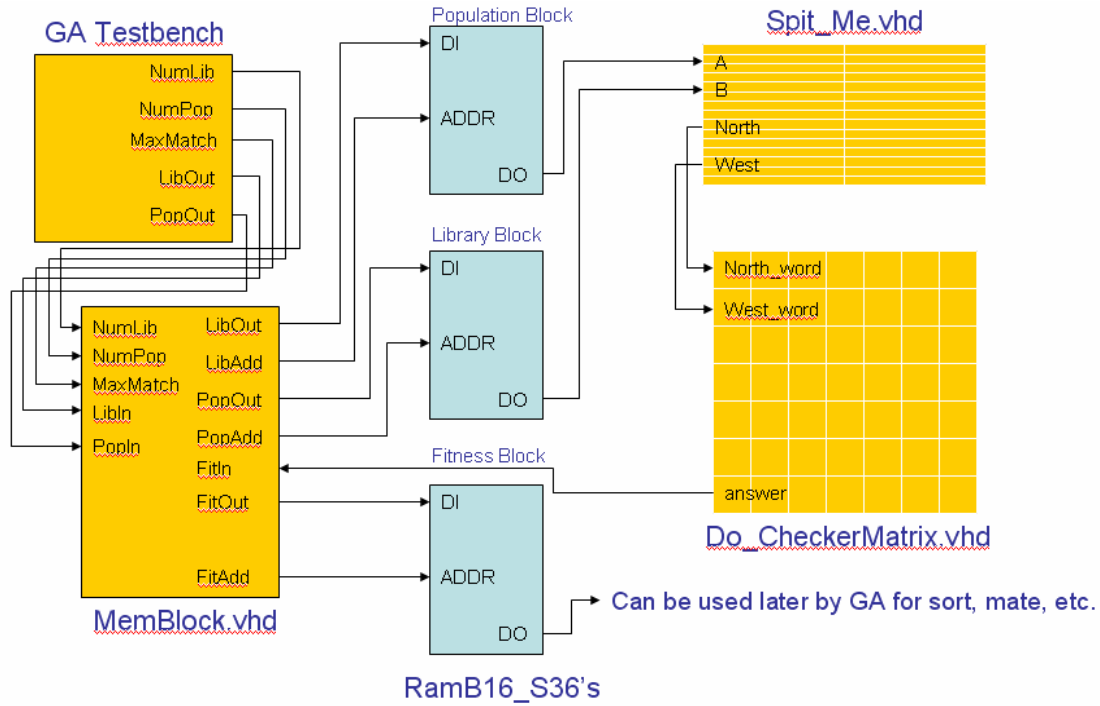


Figure 22: Upper level functional block diagram of fitness evaluator.

Table 9 shows the synthesis run report that gives information about resource utilization on the chip and expected speed of operation. This synthesis was for the case of 512 word population and word libraries, which are adequate for the runs we have made thus far. It is clear that much larger populations and libraries could be used, since only 3 of the 144 SelectBlock RAMS were used for this case. Over 80% of the chip resources are available for the GA and DNA Code Word Library application. This is encouraging, and it might even be possible to support multiple fitness evaluators, or multiple populations, on one FPGA chip.

The target speed for this application is 100MHz, since that would give us a 1000x speed-up over the software version.

Table 9 Synthesis report showing resource utilization and expected speed. Minimum period: 12.37ns (80.8MHz)

Number of Slices:	4283 out of 33792	12%
Number of Slice Flip Flops:	2544 out of 67584	3%
Number of 4 input LUTs:	7532 out of 67584	11%
Number of bonded IOBs:	98 out of 1104	8%
Number of BRAMs:	3 out of 144	2%

Actually, we know that the resource utilization show above can be cut about in half by downsizing data path width in many of the cells. The design above used 4 bit data paths in all cells, i.e. the cell can calculate values can to 16 (counting a carry out). But we know that the maximum values that will ever be computed by the cells vary through the array as shown in

Figure 23. Therefore, the cells in the top row and left column only need to be 1 bit wide, the cells in the next 2 rows and columns only need to be 2 bits wide, and so on. The ripple through version that we designed before this systolic array version did use this minimum data path sizing, and we intend to carry it into this design.

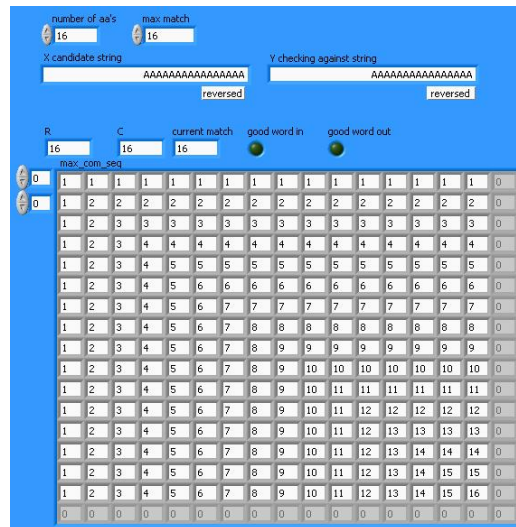


Figure 23: Maximum numbers that can be calculated in Levenshtein matrix cells.

The next higher level functional block diagram of the application is shown in Figure 24. The portions in yellow are those described above. This draft design was composed by Larry Merkle during the summer of 2003 during his Visiting Summer Faculty Research assignment in IFTC. He actually did produce synthesizable VHDL for some of the blocks, and this will serve as a starting point for us.

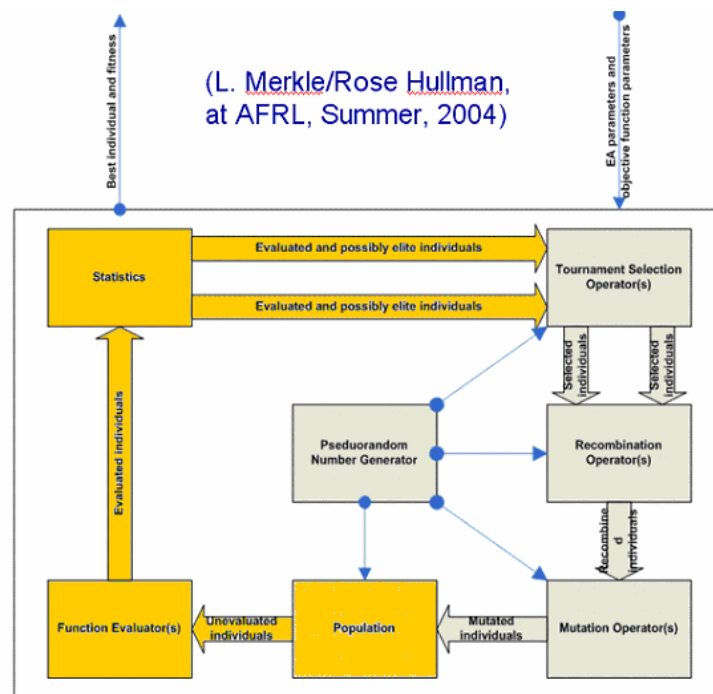


Figure 24: Higher level function block diagram of GA optimization FPGA hardware core.

Summary of work in Spiral 3

We chose to go forward into this spiral with the DNA Code Word Library Generation problem because it is an integer problem. We have investigated HOL tools for translating C to VHDL, and we have worked with one supplier's tool almost to the point of success. We hand crafted a ripple through version of the (10MHz), and also a time multiplexed systolic array version (80MHz), and we have determined that there are excellent prospects for fitting the entire application into one of our FPGA chips. We have simulated a working version of the most time intensive part of the application at about 80MHz, the fitness function evaluator, which is close to our target of 100MHz that represents a 1000x speed-up. We submitted two papers that have been accepted on the work in spiral 3, one to a FPGA meeting and one to an algorithms meeting.

Remaining work in Spiral 3

We need to complete the GA selection, mating, mutation, and random number generation blocks of the design. Then we need to integrate the GA core and DNA Code Word Library generator fitness function evaluator into one FPGA and test it. We also will evaluate doing a multiple FPGA version, which will require finding or developing support for FPGA to FPGA communications.

Future work

We have recently asked for a one year extension to this in-house effort, partly because the work in Spirals 1 and 2 were deeper and wider than we had anticipated, the workload of the PI, and because we would like to pursue additional opportunities that have arisen during the effort. One new task we would like to add is to evaluate and purchase hardware acceleration platforms


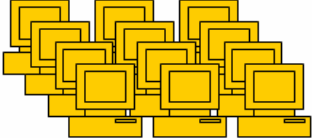



for a Notebook PC PCMCIA card platform, and prototype the GA DNA Code Word Library FPGA version on it. We expect that we will accomplish speed-ups for roughly 500x-1000x by moving the entire application to one FPGA, would be performance superior to using most clusters. We think that a notebook platform would be very attractive to users. A second additional task we would like to pursue is to evaluate and purchase FPGA board for 1 node of the AFRL/IFTC G5 BIOCOMP cluster, at to prototype the GA DNA Code Word Library application on that platform. IFTC is involved in ongoing work on that problem. A hardware accelerator could be developed for the most advanced version of the software, i.e. one that includes fuller thermodynamic constraint checking, because this can also be done in integer arithmetic.

We could also turn to accelerating other EC algorithms if that is of interest to workers in IF. We could also turn to different application problems of current interest in IF or elsewhere.

Another spin-off of this work could be to work through AFIT to develop an Application Specific Integrated Circuit (ASIC) version of the GA optimization engine. We know this is of interest to workers there in Gary Lamont's group. There are some examples of such chips, e.g. http://www.aist.go.jp/aist_e/aist_today/2002_04/2002_04_main.html#17, however, they are proprietary chips not commercially available for arbitrary applications. We know of no commercially available EC chip that could act as a hardware accelerator for optimization problems.

Finally, we are talking to a couple of labs that have developed architectures similar to this one, albeit for different types of GA, and applied to different problems. Those labs used different languages (Viva and Verilog) to develop their cores, but they might be able to provide VHDL versions for a price. We could pursue CRADA's to entice them to do this, with our side of the trade being our distributed Island Model GA that is of great interest to both labs. We have also had conversations with Virginia Tech about applying our distributed GA ODE Parameterizer to their much more complicated biological model. Presently they are focusing on a heuristic direct search method that provable will find the global minimum solution to the parameterization problem, and will contribute this to the BioSpice program. But that method requires memory to keep track of which parts of the search space have been searched, and this information grows exponentially, possibly making it unsuitable for some problems. At this point, we have acquired a free account for experimental work on their 1000 node X cluster, and we intend to at least check the speed-up of our existing distributed codes with large numbers of processors.

Figure 25 depicts a summary of the various platforms as well as current and anticipated results. It is interesting to note that the single FPGA chip platform may be far less expensive than a cluster, yet may deliver very significant speed-ups if the function evaluation can be cast into an integer systolic array.

		Speed	Resources
GA and Fitness Function (FF) on PC 0.2us GA + 9.8us FF (complete)		1	Low
GA and FF on Cluster (complete)		30X	High
GA on PC ; FF on FPGA (VHDL Synthesis Complete)		48x $10\mu s / (0.2\mu s + 10ns)^*$	Low
GA and FF on one FPGA (current work)		1,000x $(10\mu s / 10ns)^*$	Low
GA and FF on HHPC (future work)		30,000x $(30 \times 1,000x)^*$	High

* Assuming 100MHz operation

Figure 25: Speed-up and resources for the various platforms considered by this project.

Acknowledgements:

Kevin May, a Clarkson University senior, contributed greatly to the distributed GA ODE and DNA Code Word applications, and the VHDL fitness evaluator as a student intern at AFRL/IFTC during the summers of 2003-2005, and winter breaks 2003 and 2004).

Dr. Larry Merkle, Rose Hulman Institute of Technology, contributed to the GA Core for VHDL as a Visiting Summer Faculty Research Program participant in the summer of 2004.

Dr. Tony Macula, SUNY Geneseo, and Morgan Bishop, JEANSEE Corp., Geneseo, NY, contributed the Markov DNA Code Word generation results.

Dr. Ann Rundell, Purdue University, provided MatLab codes for the Ag-Ab binding model. David Pelliren/Impulse Accelerated Technologies, provided assistance in evaluating Co_Developer software.

Dr. Qinru Qiu contributed the work on the Sensor Network Energy Management Problem.

Clare Thiem AFRL/IFTC provided interface to the DARPA BIOCAMP and SIMBIOSYS programs. The DARPA SIMBIOSYS (Dr. Anantha Krisnan), BIOCAMP (Dr. Sri Kumar/DARPA), Bob Kaminski AFRL/IFGA) programs provided partial support for this work through agent fees. Dr. Thomas Renz AFRL/IFTC/BioMolecular Computing Program provided inspiration and support.

Dr. John Gallagher/Wright State University, and Prof. Gary Lamont, AFIT, provided references and valuable discussions about compact/mini-pop GAs, and about multi-objective optimization using GA, respectively.

Original Timeline and Milestones:

Quarter Calendar Year	2 03	3	4	1 04	2	3	4	1 05
Task								
1. PC software tools								
select/port GA tools for PC	x							
integrate bio-appl. w/ GA tool		x						
port to OAA, Biospice contr.			x					
GECCO 2003 EH wkshp		x						
2. Cluster tools								
select/port PC GA/bio-appl. to cluster				x				
compare PC/cluster performances					x			
to OAA, biospice cluster tool						x		
tech. paper progress (Par. Prog. Conf.)						x		
3. FPGA tools								
select/port cluster bio-appl. to FPGA						x		
ID and transition IF appl. candidates	x	x	x	x	x	x	x	x
tech. paper(s) on project results					x			x
Final Report								x

Revised Timeline and Milestones:

Quarter Calendar Year	2 03	3	4	1 04	2	3	4	1 05	2	3	4	1 06	2
Task													
1. PC software tools													
select/port GA tools for PC	x	x											
integrate bio-appl. w/ GA tool		x											
port to OAA, Biospice contr.				x									
Tech paper progress (gecco)					x								
2. Cluster tools													
select/port PC GA/bio-appl. to cluster						x							
compare PC/cluster performances						x							
to OAA, biospice cluster tool												x	
tech. paper progress (fnano, gecco, mapld)								x		x	x		
3. FPGA tools													
select/port cluster bio-appl. to FPGA									x	x	x	x	
ID and transition IF appl. candidates	x							x	x		x		
tech. paper(s) on project results													x
Interim and Final Reports											x		x

Appendix A. GECCO 2004 Paper

On Parameterizing Models of Antigen-Antibody Binding Dynamics on Surfaces – a Genetic Algorithm Approach and the Need for Speed

Daniel J. Burns¹, Kevin T. May²

¹ Air Force Research Laboratory, Information Directorate, Rome, NY 13440 USA

burnds@rl.af.mil

² Department of Computer Engineering, Clarkson University, Potsdam, NY, USA 47907

mayk@clarkson.edu

Abstract. This paper discusses the performance of a simple GA for parameterizing a particular biomodel consisting of a set of coupled non-linear ordinary differential equations. Comments are offered on the need for speed that motivates choice of language and processing platform for solving scaled problems.

1 Introduction

Ag-Ab (Antigen-Antibody) binding dynamics at surfaces is of interest to biologists because of the critical need for biosensors and diagnostic tests for the presence of targeted substances in clinical, biological, or environmental samples [Zheng 1]. Accurate models of binding dynamics are needed to support the design and performance optimization of biosensor systems. Developing accurate models requires parameterizing them to fit experimental data. This is a hard optimization problem that can easily become analytically intractable for complex nonlinear biomodels. This motivates reduced order modeling that involves techniques such as variable replacement by exogenous functions, temporal windowing, sequential parameter fitting, etc. [Rundell 2]. Even with reduced order modeling, the computing time required for parameterizations can be many minutes. The present work evaluated whether a simple GA approach run on a full, unreduced model could be more efficient. This work served as a test case for developing working C codes that could be ported to parallel and embedded computer platforms to achieve extreme speed-ups known to be needed for certain hard problems.

2 Summary of the work and results

We wrote codes for simulating the Model and for parameterizing the Model with a simple GA in Labview and in MatLab, and translated them to C to achieve reasonable run times (100-10,000 X faster). Our GA used binary representations of the floating point Model parameters. Values were searched over adjustable width ranges that were different for each parameter. Adjustable GA parameters included # bits in each gene, # chromosomes, # genes in each chromosome, # in initial population, # in running population, % of population selected to produce children, % of gene bits mutated at each generation, maximum # generations, ranking method for selection (probability based on fitness or rank), and termination criteria (based on total fitting error, or maximum single point error). We used uniform probability crossover for all genes, and elitism (keeping the current best individual).

The Model equations predict the # of Ag particles bound to an Ab functionalized surface, with one equation describing each attachment *valency* (i.e. the # of epitope sites binding an Ag particle). The Model has 6 parameters (initial association rate, initial dissociation rate, transport coefficient due to diffusion, transport coefficient due to gravity, crosslinking association rate, and crosslinking dissociation rate). A *complete* raw data set from the Model consisted of about 7 concentration vs. time curves tracing out a binding and release experiment over 200 time points. Experimentally it is only possible to measure the *total* # of bound Ag, not the *valency* of a bound Ag. Therefore, we summed the curves to produce a single *total bound* concentration vs. time curve. We measured the performance of the GA parameterization tool working with both *complete* data sets and reduced *total bound* data sets. The fitness function calculated the sum of the least squares differences of data sets from a *known* individual (with a set of preselected parameter values) and individuals in the population. Various supervisory programs were written to

gather statistical performance results over typically 20 or 30 fitting runs, and to display results. Also, a number of metrics were defined to characterize the performance of the GA fitter (speed, accuracy, convergence) using different sets of GA parameters, and for determining how the metrics behaved with scaling, e.g. as a function of population sizes, maximum generations, etc.

We observed that fitting all six parameters to about 2% accuracy using *complete* data sets was easy using populations of about 200 run for 1000 generations. This involved about 200,000 function evaluations and took about 30 seconds. Times for fitting 5 parameters (without kg which hardly affects the data) were much better, typically about 2 seconds. We also determined that varying the ranges over which parameter values were fit (from 2 to 4 to 10 times the parameter values) changed the average number of generations required to converge from 500 to 800 to 1250.

Fitting all 6 Model parameters simultaneously from *total bound* data sets proved to be much harder, with typically only 10% of runs achieving <10% fitting errors. We are currently experimenting with a progressive GA (PGA) parameter fitting strategy that mimics the progressive fitting strategy described in [2] to improve competence.

References

1. Zheng, Y.; Rundell, A., "Biosensor Immuno-surface Engineering Inspired by B-cell Membrane Bound Antibodies: Modeling and Analysis of Multivalent Antigen Capture by Immobilized Antibodies, IEEE Transactions on NanoBioscience, 2(1):14-25, 2003.
2. Rundell, A.; DeCarlo, R.; Doerschuk, P.; HogenEsch, H.; "Parameter Identification for an Autonomous 11th Order Nonlinear Model of a Physiological Process", Proceedings of the 1998 American Control Conference, 6: 3585-3589, 1998.



On Parameterizing Models of Antigen-Antibody Binding Dynamics on Surfaces:

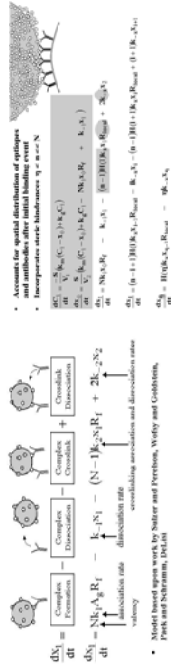
A Genetic Algorithm Approach and the Need for Speed
 Daniel J. Burns, Air Force Research Laboratory, Information Directorate,
 Kevin N. May, Clarkson University, mayk@clarkson.edu

burnds@rl.af.mil

Abstract: This paper discusses the performance of a simple GA for parameterizing a test case bio-model consisting of a set of coupled non-linear ordinary differential equations describing Ag-Ab binding dynamics on surfaces. Solutions were written in three software environments and compared.

The test case Bio-Model:

- Ag-Ab binding dynamics on surfaces
- relevant to critical need for biosensors and diagnostics
- focus of one project in a current DARPA program



Results:

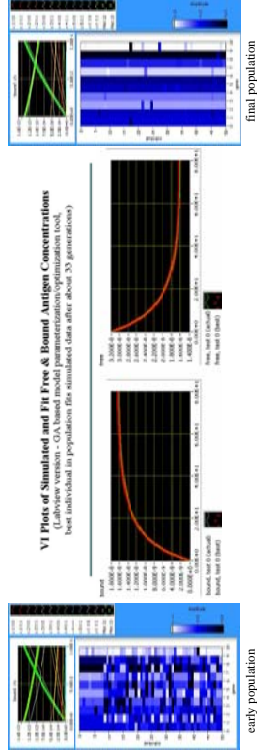
- fitting based on available non-GA optimization tool box - did not converge using full model
- can do fitting with non-GA tool box using reduced order model and data windowing
- GA fits full model easily with full data sets
- ported to C for reasonable speed with human operator

Sponsors:



Collaborators:

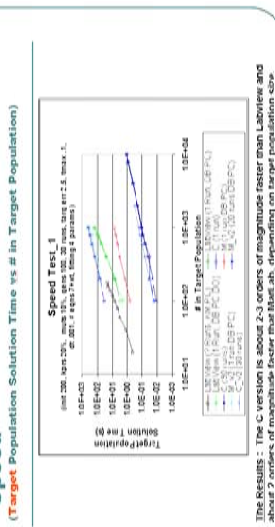
A. Rundell/Purdue, G. Lamont/AFIT, L. Merkle/Rose-Hulman



GA/Model fitter:

- real valued double precision model parameters scaled to integer gene values for GA
- elitism, uniform crossover, bit flip mutations
- least squares difference fitness function
- termination criteria: # generation & worst fit point

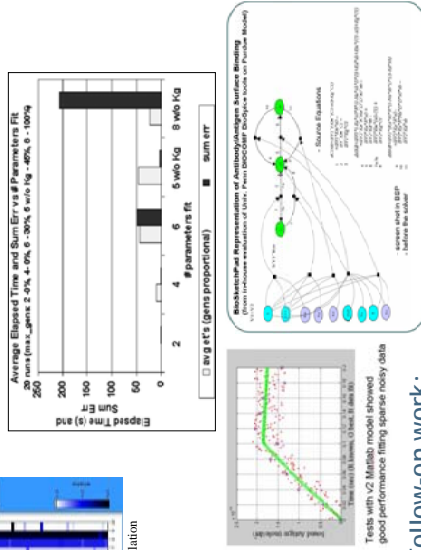
Performance Evaluation Results



References:

1. Zheng, Y.; Rundell, A., "Biosensor Immuno-surface Engineering Inspired by B-cell Membrane Bound Antibodies: Modeling and Analysis of Multivalent Antigen Capture by Immobilized Antibodies, IEEE Transactions on Nanobioscience, 2(1):14-25, 2003.
2. Rundell, A.; DeCarlo, R.; Doerschuk, P.; HogenEsch, H.; "Parameter Identification for an Autonomous 11th Order Nonlinear Model of a Physiological Process", Proceedings of the 1998 American Control Conference, 6: 3585-3589, 1998.

- Run Managers & Characterizations:**
- always measured statistics over ~30 runs
 - checked speed, convergence, accuracy
 - varied problem difficulty (# par., range widths)
 - checked incomplete/noisy/sparse data sets



Follow-on work:

- Porting to open agent architecture Bio-Spice environment
- Porting to FPGA, cluster, and heterogeneous cluster for speed (MSAEC poster paper GECCO 2004).
- apply to other problems of interest: multi-objective optimization, planning, scheduling, ATR, network & distributed database design and operation, Bayesian network composition, hyperspectral imaging analysis, antenna design, topics in computing and sensing with biology, dna code word library synthesis

Appendix B. FNANO 2005 Paper

DNA Code Word Library Generation Using A Parallel Genetic Algorithm

Dan Burns and Kevin May, Air Force Research Lab, Morgan Bishop, JEANSEE Corp.,
Geneseo, NY.

The composition of DNA code word libraries useful for data storage, tethering to surfaces, and self assembly applications is computationally intensive, and the time required to discover large libraries increases as the number and complexity of constraints aimed at controlling strand interaction increase. This poster paper describes a parallel Genetic Algorithm (GA) for generating such libraries on a cluster of computers, and compares its performance to that of Markov and Stochastic methods (see references on poster).

The parallel GA utilizes multiple computer nodes in a cluster that effectively compete to add the next good word pair to the library. Each node evolves candidates in separate or “island model” populations that are different on each node. However, communication among the nodes provides for sharing of new words found on one node by all nodes, and also for “diffusion” of a few highly fit, but not yet “good” words from each node to the next node around a ring loop. The method can be adapted to any set of constraints.

We tested the performance of the GA relative to a Markov method for the case of composing libraries that satisfy both a minimum edit distance, measured by the Levenstein matrix (Lv) and reverse code, i.e. both code words and their reverse compliments are present (RC), constraints. Both methods performed similarly in terms of the rate of finding words up to the point in time where the search becomes “difficult”. We did observe that the Markov method was able to discover more words (131 pairs vs. 115 for a length 16, edit distance 10, Lv RC code, although at this writing we are not certain whether this is due to longer total run time or the effects of early choices of words for the library. (This point will be addressed before the meeting). Speed-up curves show that both methods scale about linearly over the range of 1-30 processor nodes used.

We tested the performance of the GA relative to a Stochastic method, for the cases of composing both 16/10 Lv RC codes and 8/4 Hamming Distance (HD) HD RC codes. The GA is significantly faster up to the point where the search becomes “difficult”, because it starts with an empty library and breeds candidates to possibly add, whereas the Stochastic method starts with a full library and does mutations to improve constraint satisfaction. Thus GA runs far fewer “checks” of constraints than Stochastic.

Finally, execution time profiling shows that most of the total time is spent evaluating the Levenstein matrix, regardless of the method used (e.g. 98%+ for the GA). Since this is an integer-only calculation, it could be implemented in an embedded FPGA. We intend to do this in future work, and the poster will consider some details of this approach and estimate expected the speed-up.



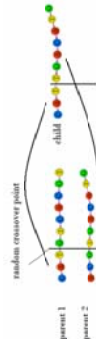
Parallel Genetic Algorithm for DNA Codeword Library Design

Daniel J. Burns, Air Force Research Laboratory, Rome, NY burnsd@af.mil, Kevin May, Clarkson College & AFRL, Potsdam, NY, maykn@clarkson.edu, Morgan Bishop, JEANSEE Corp., Geneseo, NY, bishopm@rl.af.mil

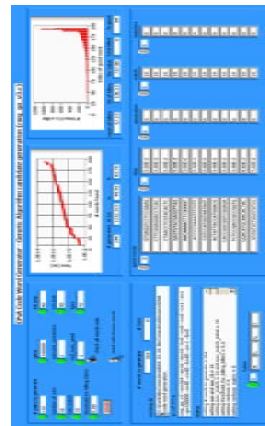


Genetic Algorithm Code Word Generator details:

Population size: 100 per processor
Selection for mating: based on fitness (population is sorted)
Fitness Function: (# rejecters) and (match-max_match)
Mating: 2 parent single point cross-over, probability 0-20%



Mutation: best possible single base mutation, probability 1%
Clones: not allowed
Termination Criteria: max_gens, max_time, all words found



Problem:

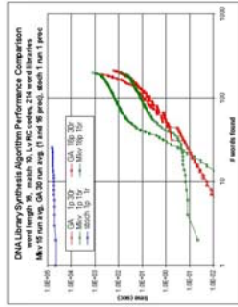
Generation of large DNA code word libraries is compute intensive. Design time increases significantly for larger libraries of longer words with more difficult constraints.

Objective:

Evaluate alternative algorithms, parallel codes, and potential embedded (FPGA) approaches for achieving speed-ups.

Approach:

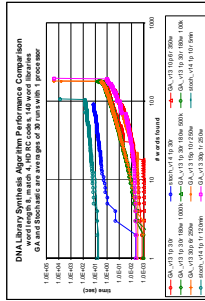
Implement a distributed Genetic Algorithm code word generation tool and compare its performance with that of Markov and Stochastic methods.



Code Types:
 LMI Levenshtein Matrix
 RC reverse complement
 HD Hamming Distance

Results:

- Markov and Genetic Algorithms perform similarly building 16/10 library of 214 words, checking only Levenshtein Matrix and Reverse Complement constraints. Both start with empty libraries.
- Stochastic algorithm run time is sensitive to initial random library size and content (starts with filled random library)

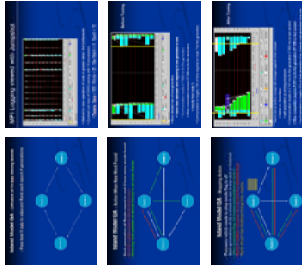


Stochastic Algorithm Code Word Generator details:

- based on Tulpan, Hoos, Condon, 2002.
- initial library of random candidate words improved by mutations
- low probability (20%) of random single base mutations (if beneficial)
- high probability (80%) of best possible single base mutations (if beneficial)

A. Modina, W. Ropasch, V. Rytov, M. Bishop, "DNA Codeword Library Design", FNANO 2005, April 2005.
 R. Deaton, R. C. Murphy, J. A. Rose, D. R. Franceschetti, and S. E. Stevens, "Genetic search of reliable encodings for DNA-based computation", Proc. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 44, 1999, pages 247-258.
 R. Deaton, M. Garzon, R. C. Murphy, J. A. Rose, D. R. Franceschetti, and S. E. Stevens, "Genetic search of reliable encodings for DNA-based computation", Proc. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 44, 1999, pages 247-258.
 D.C. Tulpan, H. Hoos, A. Condon, "Stochastic Local Search Algorithms for DNA Word Design", Eighth International Meeting on DNA Based Computation (DNAB), June 2002.
 Arvind Brennen and Anne E. Condon, "Strand Designer for Bio-Molecular Computation", Theoretical Computer Science, Vol. 257, Issue 1, Sept. 2002, Natural computing, Pages 39 - 58.

Parallel version details: Island model GA. Each processor evolves a population, but interacts in 3 ways.



Drift 5 best individuals:

Best 5 individuals of each population are migrated to an adjacent processor or at the end of 40 generation epochs

Same library on all nodes:

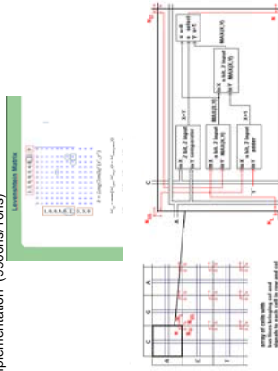
new words are shared with all populations when found

first processor to finish wins

Speed-up: Approximately linear for GA LRC code 16/10, 200 words, avg of 15 runs, 15-30 processors (run with mpi run -np 30 CWG GA v9 -a=16 -e=40 -q=100000 -l=30000 -k=3000 -m=10 -r=3000 -s=1 -w=200 -z=1



Future Work: Time profiling shows that about 97%+ of the computation time while composing LRC code libraries is spent evaluating the Levenshtein matrix. This calculation involves only integers, with ~ 1000X speed-up projected for FPGA implementation (990ns/10ns)



Appendix C. GECCO 2005 Paper

DNA Code Word Library Generation Using A Parallel Genetic Algorithm

Dan Burns
Air Force Research Laboratory
525 Brooks Road
Rome, NY 13441-4505
315-330-2335
burnsd@rl.af.mil

Kevin May
Clarkson College & AFRL
26 Electronic Parkway
Rome, NY 134441-4514
315-330-2335
maykn@clarkson.edu

Morgan Bishop
JEANSEE, Corp.
Geneseo, NY
315-330-1556
Morgan.Bishop@rl.af.mil

ABSTRACT

DNA code word libraries are useful for implementing data storage and computation schemes involving concatenated words with ‘sticky ends’ or tiles joined in predictable patterns by means of ‘sticky edges’, for tethering bio-molecules to surfaces in diagnostic and sensor applications, as well as for self assembly of nano-scale templates that may serve as precursors for arraignment of other nano-scale devices. Composing such libraries is computationally intensive, with the time required to discover large libraries increasing as the number and complexity of constraints aimed at controlling strand interaction increases. This poster paper describes a parallel Genetic Algorithm (GA) for generating such libraries on a cluster of computers, and compares its performance to that of Markov and Stochastic methods. A hardware implementation is proposed for speeding up the time consuming Levenstein Matrix constraint checking calculation.

Categories and Subject Descriptors

D.1.3 [**Software**]: Programming Techniques – concurrent programming – *Parallel Programming*

F.2.3 [**Theory of Computing**]: Computation by Abstract Devices – Nonnumerical Algorithms and Problems – *sorting and searching*

G.1.6 [**Mathematics of Computing**]: Numerical Analysis – Optimization – *constrained optimization*

General Terms

Algorithms, Performance, Design, Experimentation.

Keywords

DNA, codes, word, library, genetic

EXTENDED ABSTRACT

The composition of DNA code word libraries useful for data storage, tethering to surfaces, and self assembly applications is computationally intensive, and the time required to discover large libraries increases as the number and complexity of constraints aimed at controlling strand interaction increase. This poster paper describes a parallel Genetic Algorithm (GA) for generating such libraries on a cluster of computers, and compares its performance to that of Markov and Stochastic methods (see references on poster).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GECCO’05, June 25-29, 2005, Washington, DC, USA.
Copyright 2005 ACM 1-59593-010-8/05/0006...\$5.00.

The parallel GA utilizes multiple computer nodes in a cluster that effectively compete to add the next good word pair to the library. Each node evolves candidates in separate or “island model” populations that are different on each node. However, communication among the nodes provides for sharing of new words found on one node by all nodes, and also for “diffusion” of a few highly fit, but not yet “good” words from each node to the next node around a ring loop. The method can be adapted to any set of code word constraints.

We tested the performance of the GA relative to a Markov method for the case of composing libraries that satisfy both a minimum edit distance, measured by the Levenstein matrix (Lv) and reverse code (RC), i.e. both code words and their reverse compliments are present, constraints. Both methods performed similarly in terms of the rate of finding words up to the point in time where the search becomes “difficult”. We did observe that the Markov method was able to discover more words (131 word pairs vs 115) for a length 16, edit distance 10, Lv RC code, although at this writing we are not certain whether this is due to longer total run time or the effects of early choices of words for the library. (This point will be addressed before the meeting). Speed-up curves show that both methods scale about linearly over the range of 1-30 processor nodes used.

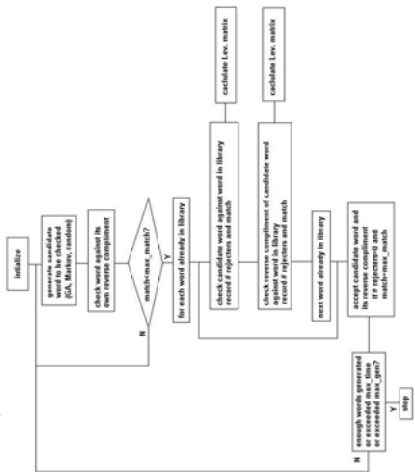
We tested the performance of the GA relative to a Stochastic method, for the cases of composing both 16/10 Lv RC codes and 8/4 Hamming Distance (HD) RC codes. The GA is significantly faster up to the point where the search becomes “difficult”, because it starts with an empty library and breeds candidates to possibly add, whereas the Stochastic method starts with a full library and does mutations to improve constraint satisfaction. Thus GA runs far fewer “checks” of constraints than Stochastic.

Finally, execution time profiling shows that most of the total time is spent evaluating the Levenstein matrix, regardless of the method used (e.g. 98%+ for the GA). Since this is an integer-only calculation, it could be implemented in an embedded FPGA. We intend to do this in future work, and the poster will consider some details of this approach and estimate expected the speed-up.



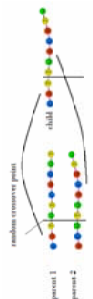
DNA Codeword Library Design Using A Parallel Genetic Algorithm

Daniel J. Burns, Air Force Research Laboratory, Rome, NY bunsd@rl.af.mil, Kevin May, Clarkson College & AFRL, Potsdam, NY, maykn@clarkson.edu, Morgan Bishop, JEANSEE Corp., Geneseo, NY, bishopm@rl.af.mil

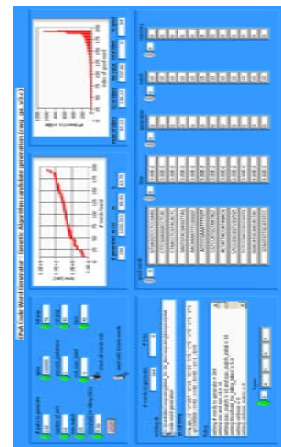


Genetic Algorithm Code Word Generator details:

Population size: 100 per processor
 Selection for mating: based on fitness (population is sorted)
 Fitness Function: (# rejecters) and (match-max_match)
 Mating: 2 parent single point cross-over, probability 0-20%



Mutation: best possible single base mutation, probability 1%
 Clones: not allowed
 Termination Criteria: max_generations, max_time, all words found



Problem:

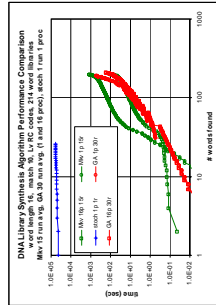
Generation of large DNA code word libraries is compute intensive. Design time increases significantly for larger libraries of longer words with more difficult constraints.

Objective:

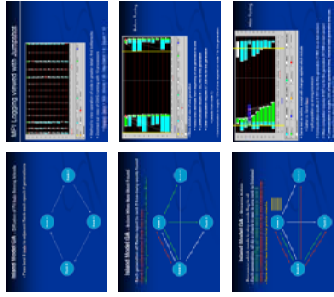
Evaluate alternative algorithms, parallel codes, and potential embedded (FPGA) approaches for achieving speed-ups.

Approach:

Implement a distributed Genetic Algorithm code word generation tool and compare its performance with that of Markov and Stochastic methods.



Code Types:
 LVM Levenshtein Matrix
 RC reverse complement
 HD Hamming Distance



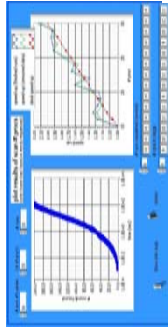
Parallel version details: Island model GA. Each processor evolves a population, but interacts in 3 ways.

best 5 individuals of each processor's population are migrated to an adjacent processor every 40 generation epoch

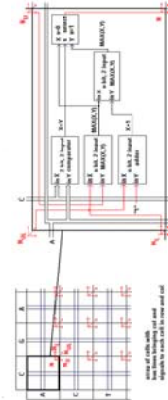
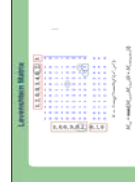
new words are shared with all populations when found

first processor to finish wins

Speed-up: Approximately linear for GA LVM code 16/10, 200 words, avg of 15 runs, 15-30 processors (run with mpirun -np 30 cwg_ga_v9 --a=16 --e=40 --g=100000 --l=3000 --k=3000 --m=10 --r=3000 --s=1 --w=200 --z=1)



Future Work: Time profiling shows that about 97%+ of the computation time while composing LV RC code libraries is spent evaluating the Levenshtein matrix. This calculation involves only integers, and can be implemented in an FPGA.



A. Marcia, W. Popowicz, V. Rykov, M. Bishop, "DNA Codeword Library Design", FINANO 2005, April 2005.
 R. Deaton, R. C. Murphy, M. J. Condon, D. R. Francischetti, and S. E. Stevens, Jr., "Good encodings for DNA-based solutions to the combinatorial search problem", DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 44, 1999, page 247-258.
 R. Deaton, M. Garzon, R. C. Murphy, J. A. Rose, D. R. Francischetti, and S. E. Stevens, Jr., "Genetic search of reliable encodings for DNA-based computation", Kozs, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors), Proceedings of the First Annual Conference on Genetic Programming 1996. (6 words of 2006)
 D.C. Tulpan, H. Hoos, A. Condon, "Stochastic Local Search Algorithms for DNA Word Design", Eighth International Meeting on DNA Computing and Molecular Computation, Theoretical Computer Science, Vol. 297, Issue 1, Sept. 2002, Natural computing, Pages: 39 - 58.

Stochastic Algorithm Code Word Generator details:

- based on Tulpan, Hoos, Condon, 2002.
 - initial library of random candidate words improved by mutations
 - low probability (20%) of random single base mutations (if beneficial)
 - high probability (80%) of best possible single base mutations (if beneficial)